Universidade Nova de Lisboa

Faculdade de Ciências e Tecnologia

Departamento de Informática

Integrando Problem Frames com Aspectos

Por

Gustavo Salvador Marquês

Nº 26498

Dissertação apresentada na Faculdade de Ciências e Tecnologia
da Universidade Nova de Lisboa para
obtenção do grau de Mestre em Engenharia Informática

Orientador

Prof. Doutor João Baptista da Silva Araújo Júnior

Co-Orientadora

Prof. Doutora Maria Lencastre

Monte da Caparica

2009

Agradecimentos

Antes de mais, gostaria de agradecer todo o apoio que me foi dado por todos os meus familiares, que contribuíram e possibilitaram a conclusão de mais uma fase crucial da minha vida. Desde os meus pais e irmã, os meus avós, à minha tia e claro os meus padrinhos. Fizeram os possíveis e impossíveis.

Aos meus amigos mais chegados, Ivo e sua mãe, João e seus pais, Miguel e Vicente que estiveram sempre presentes ao longo destes anos e que tanto me ajudaram. Foram, são e sempre serão os únicos que dão verdadeiro significado à palavra amizade.

Não posso esquecer os meus colegas de faculdade, que providenciaram um fantástico grupo de trabalho durante os últimos anos de curso e que sem eles nada teria sido realizável. Por isso, os meus sinceros agradecimentos ao Marco, Lobo, Domingues, André R., Daniel, Hugo, David, Pedro, e mais recentemente Ricardo, André S. e Filipe.

Uma das fases mais importantes deste percurso foi a minha primeira experiência profissional. Na TMN, colegas como o Hugo, Valério e Igor, que desde então têm estado presentes, não podem nunca ser esquecidos.

Por último, mas não menos importante, o meu especial agradecimento aos meus incansáveis orientadores, o professor João Araújo, e a professora Maria Lencastre, por todo o apoio, dedicação e esforço. A conclusão deste Mestrado apenas foi possível graças a eles.

Um grande obrigado a todos!

Resumo

A primeira actividade do processo de desenvolvimento de software consiste na identificação das funcionalidades e propriedades que se pretendem ver implementadas nos sistemas, sendo designada por engenharia de requisitos.

Nesta fase de análise de requisitos podem surgir problemas que, se não forem correctamente solucionados, irão posteriormente conduzir a um agravamento de custos num estágio mais avançado do desenvolvimento. Um exemplo disto são elementos que se encontram espalhados e repetidos por diversas componentes do sistema que se pretendem modularizar, designados por assuntos transversais (*crosscutting concerns*), e que afectam o seu comportamento (aspectos).

Os mecanismos de Engenharia de Requisitos Orientada a Aspectos, dão suporte a que este tipo de conjunturas seja minimizado, fornecendo maneiras de lidar de forma eficaz com estes *concerns*.

A abordagem "Problem Frames" é uma técnica de análise de requisitos bastante conhecida e estabelecida na comunidade de requisitos, composta por vários conceitos com o objectivo de reunir e especificar as funcionalidades de um problema em engenharia de software, bem como reutilizar conhecimentos relacionados a classes de problemas previamente conhecidas. No entanto, este método pode ainda tornar-se bastante mais poderoso, isto se permitir lidar com elementos transversais, através da integração de noções orientadas para aspectos.

Abstract

In the software development process, its first activity consists at identifying the features and properties to be implemented in any kind of systems, which it is called requirements engineering.

At this point of requirements analysis, many problems may arise, and if they are not properly resolved, it will subsequently lead to higher costs in a more advanced stage of the development. An example of this are some elements that are distributed and repeated by several components of the system, which should be modularized, also designated by crosscutting concerns, and that affect their behavior (aspects).

The mechanisms of Aspect-Oriented Requirements Engineering ensure that such situations will be minimized by providing ways to deal with these concerns effectively.

The “Problem Frames” approach is a requirements analysis technique very well known and established in the requirements community, consisting of several concepts, in order to gather and specify the functionality of a software problem and reuse knowledge related to previously known classes of problems. However, this method can become much more powerful if allowed to deal with crosscutting concerns, particularly regarding to the integration of aspect-oriented notions.

Acrónimos

Acrónimo	Significado
AORA	Aspect Oriented Requirements Analysis
AORE	Aspect Oriented Requirements Engineering
AOSD	Aspect Oriented Software Development
ARCaDe	Aspectual Requirements Composition and Decision
EROA	Engenharia Requisitos Orientada a Aspectos
IPS	Interaction Pattern Specification
KAOS	Knowledge Acquisition in Automated Specification
MATA	Modeling Aspects using a Transformation Approach
OCL	Object Constraining Language
PF	Problem Frames
PREView	Process and Requirements Viewpoints
UML	Unified Modeling Language
VAODA	Viewpoint and Aspect Oriented Domain Analysis Approach
VORD	Viewpoint Oriented Requirements Definition
XML	eXtensible Markup Language

Índice de Matérias

Índice de Figuras

Índice de Tabelas

1. Engenharia de Software

1.1. Engenharia de requisitos

Na área de engenharia de software, para que o desenvolvimento e manutenção do software sejam efectuados de uma forma disciplinada, é indispensável a utilização de métodos que garantam a integridade das aplicações. Nas fases iniciais do desenvolvimento de software, todas as actividades relacionadas são objecto de estudo da disciplina de engenharia de requisitos.

Esta consiste, principalmente, na análise de todos os serviços que o sistema deve providenciar, bem como nas restrições operacionais a ele associadas, procurando descobrir, documentar e testar todas as suas funcionalidades [27].

Na análise de requisitos, é necessário inicialmente comunicar com os clientes, a fim de determinar quais as funcionalidades pretendidas (através de entrevistas, observação de tarefas, entre outras). Nesta fase, tornam-se indispensáveis a descrição e documentação de cada requisito, de modo a que sejam úteis no decorrer do desenvolvimento. A engenharia de requisitos é caracterizada pelas seguintes actividades [27]:

- *Elicitação e análise de requisitos* – processo de interacção com os clientes (pessoa ou grupo afectado directa ou indirectamente pelo sistema) para recolha de informação sobre o sistema proposto, diferenciando os requisitos do utilizador (serviços que se esperam ver implementados e restrições que o afectam) e de sistema (funções e limitações operacionais que definem exactamente o que deve ser desenvolvido). Esta actividade requer ainda a classificação, organização e documentação dos requisitos, que por sua vez pode ser realizada através de linguagem natural, diagramas de modelo ou especificações formais.
- *Validação de requisitos* – Este passo garante que os requisitos definem correctamente o sistema pretendido pelo cliente. Assume grande importância, tendo em conta que os custos e insatisfação derivados à presença de erros, isto após a aplicação já se encontrar em funcionamento, são deveras maiores que a sua resolução na fase de desenvolvimento.

- *Gestão de requisitos* – Responsável por compreender e controlar as possíveis mudanças das funcionalidades do sistema, garantindo a conformidade das dependências entre estas.

Esta dissertação incide principalmente no primeiro ponto, onde podem ser utilizadas várias abordagens que permitem a identificação e especificação de cada requisito, seja este funcional (relacionado com a operacionalidade ou funcionalidade que o sistema deve suportar) ou não funcional (restrição sobres os serviços ou atributos de qualidade da aplicação).

Os requisitos funcionais [27, 9] pretendem representar como o sistema se deve comportar em certas situações e como este deve reagir a certas acções. Requisitos não funcionais [27, 9] estabelecem regras ou restrições que se aplicam às operações providenciadas pelo sistema, podendo afectar várias componentes deste, tais como o tempo de resposta, segurança ou disponibilidade.

Em cada etapa do desenvolvimento de software, dever-se-á decidir e escolher qual a técnica mais adequada a utilizar para a aplicação que se encontra a ser desenvolvida. Na base desta decisão estão factores tais como, os requisitos do sistema, práticas organizacionais, restrições, ou até mesmo os requisitos transversais, do inglês, *crosscutting requirements* (requisitos aspectuais de um sistema que afectam e se encontram espalhados em diversas componentes deste) [6].

Entre algumas destas técnicas encontram-se abordagens orientadas a objectivos (*goal-oriented*), em que se designam por *goals* os fins que se pretendem ver alcançados, tais como o KAOS [9], e i* Framework [9].

Outros métodos são baseados em: *viewpoints* (representam perspectivas diferentes do sistema global) como PREView [26, 27]; cenários [27]; casos de uso (*use cases*) [14, 27]; ou padrões de problemas como no caso de Problem Frames [8, 13, 9, 16]. Mais recentemente temos as abordagens orientadas a aspectos [6, 19], que modelam o sistema tendo em conta a separação e modularização de requisitos que se encontram espalhados ou afectam várias partes do sistema (i.e. requisitos aspectuais).

Neste trabalho o maior foco estará, no entanto, centrado no método de Problem Frames [13] (categoriza problemas no desenvolvimento de software), integrando-o com a técnica AORE [22] (*Aspect-Oriented Requirement Engineering*) baseada em *viewpoints*.

1.2. Motivação

Várias abordagens tradicionais de engenharia de requisitos já providenciam meios para a detecção, especificação e composição em módulos de *crosscutting requirements*, durante as fases preliminares do crescimento do software. Se assim não fosse, os requisitos aspectuais continuariam a encontrar-se distribuídos pelas diversas componentes do sistema, impedindo a sua implementação de uma forma mais modularizada, com uma estrutura mais compreensível e fácil de modificar. Entretanto pouco se tem feito a fim de integrar os conceitos de aspectos em Problem Frames.

1.2.1. Crosscutting concerns

O principal foco de *Aspect-oriented software development* (AOSD) reside no modo como modularizar e tratar de forma eficaz cada assunto transversal, do inglês, *crosscutting concern*. Um *concern* é um assunto, um conceito que se deve ter em conta num sistema. Um *concern* pode ser refinado ou descrito por um ou mais requisitos.

No âmbito da engenharia de requisitos, um caso particular de requisito é designado por requisito aspectual ou *crosscutting requirement*. Independentemente da sua natureza funcional, estes são caracterizados por possuírem um tipo de relacionamento especial com outros requisitos. Estes podem descrever diversas características funcionais ou não funcionais do software, que afectam outras no decorrer do seu desenvolvimento, encontrando-se dispersas e repetidas por vários módulos do sistema, dificultando a sua compreensão, manutenção e evolução em fases futuras.

Quanto mais cedo for realizada a identificação e documentação deste tipo de requisitos, menor será o seu impacto negativo no desenvolvimento de um sistema. No entanto, existem diversas fases do ciclo de vida do software em que é possível efectuar este processo [24], tais como, durante e após a modelação de requisitos e sua especificação, ou até mesmo, através de actividades mais adiantadas no procedimento de manutenção.

Através das diversas técnicas centradas em Engenharia de Requisitos Orientada a Aspectos (EROA), é possível providenciar meios de identificar, separar, representar e compor *crosscutting concerns*, para que estes possam ser encapsulados em módulos independentes, isto claro, numa fase inicial do processo de desenvolvimento do software (ao nível da elicitação de requisitos), passando a designar-se como aspectos [6].

Um aspecto permite não só realizar a modularização de *crosscutting concerns*, que anteriormente não seria possível, obtidos por intermédio de um simples *use case* ou *viewpoint*, mas também auxiliar o programador, facilitando a identificação de conflitos, permitindo, assim, tornar os custos de desenvolvimento e manutenção de *software* mais reduzidos. Portanto, a melhor maneira de lidar com requisitos aspectuais é separá-los e modularizá-los o mais cedo possível.

1.2.2. Problem Frames e Crosscutting concerns

De modo a possibilitar um maior conhecimento sobre Problem Frames, tomar-se-á como ponto de partida as noções introduzidas por Michael Jackson em [13, 12], com vista a uma melhor compreensão e descrição de problemas.

Em engenharia de requisitos de software, a principal característica que distancia esta abordagem de muitas outras, é a importância dada à separação do problema e solução.

Desta forma, pretende-se demonstrar que, para se dar início à análise de um problema, é necessário previamente identificar e localizar todas as componentes a ele pertencentes, bem como, definir todas as funcionalidades que deverão estar presentes. Assim, surge a necessidade da sua modelação em forma de diagramas, complementados por descrições (textuais, formais ou informais). Para se atingir eficientemente este pressuposto, terá de se conseguir determinar o modo como os diversos domínios que constituem o problema se encontram interligados e interagem entre si, aliando-se a dois tipos de descrições, indicativas (correspondem aos domínios físicos, pertencentes ao mundo real, e à forma como comunicam por intermédio de fenómenos) ou optativas (restrições respeitantes aos comportamentos do sistema).

No entanto, no que toca à técnica de Problem Frames [13], esta ainda se encontra numa fase de aperfeiçoamento, especialmente no que concerne à definição de requisitos não funcionais e no que toca à representação e modularização de elementos *crosscutting*. Deste modo, é absolutamente necessário melhorar os seus modelos para que se possibilite a integração de conceitos orientados a aspectos, providenciando também formas e métodos de identificar e resolver *concerns* não funcionais conflituosos. Isto garantiria que o processo de elicitação de requisitos se tornasse bastante mais completo na sua análise e descrição do problema, permitindo uma maior aposta na sua utilização no mundo da engenharia de software, pois é uma prática ainda muito pouco considerada para estes fins.

1.3. Objectivos

Neste momento, para além de o principal objectivo de Problem Frames [13] ter o seu foco no ambiente onde se encontra situado o problema, existem já avanços e propostas que expandem um pouco o modo como esta técnica é modelada com aspectos [18], definindo relações entre elementos do seu modelo, ou até mesmo derivando propriedades não funcionais.

O trabalho desta dissertação tem como principal meta visar um processo de captura e integração de aspectos no modelo de análise de Problem Frames [13], que garanta uma óptima modularização e estruturação de todos os requisitos obtidos por intermédio da investigação de um problema de software.

De forma a atingir o objectivo proposto, a decisão sobre qual a técnica que melhor integra a abordagem de Problem Frames [13] teve como ponto de partida, o estudo de diversas abordagens EROA existentes. Esta escolha baseou-se nas características de cada processo, sendo que aquela que melhor se enquadrou neste contexto foi o método AORE [22].

Isto deve-se, em grande parte, à maturidade e objectividade evidenciada por este método, no que diz respeito à identificação e especificação de requisitos através da utilização de *viewpoints*, bem como ao suporte da separação de propriedades *crosscutting* (não funcionais) e resolução de conflitos daí emergentes.

A abordagem proposta nesta dissertação tem como finalidade a elaboração de um modelo de análise e tratamento de requisitos, que integre as abordagens Problem Frames [13] e AORE [22], aplicando tarefas que visem tirar proveito das principais características de cada um dos métodos. Desta forma, as contribuições deste trabalho são as seguintes:

- Integrar a abordagem de Problem Frames [13] com a abordagem AORE [22], com o fim de identificar aspectos mais cedo para obter modelos de *problem frames* mais modularizados.
- Definir e aplicar regras de composição entre os aspectos e os elementos por eles afectados.
- Melhorar a forma de organizar os subproblemas, através da utilização de *viewpoints*, requisitos e funcionalidades do problema.
- Definir um meta-modelo para AORE [22].
- Relacionar os meta-modelos de Problem Frames [11] e AORE [22].

1.4. Organização

Esta dissertação contém cinco capítulos. O primeiro capítulo é esta introdução, em que é introduzido o tema deste trabalho e o contexto em que se encontra inserido, assim como, são apresentadas também as principais razões e a motivação central desta tese, juntamente com os objectivos que se propõe alcançar. Os outros capítulos são abaixo mencionados:

- Capítulo 2. Problem Frames e Engenharia de Requisitos Orientada a Aspectos. Neste capítulo são apresentados as abordagens relacionadas com o âmbito do trabalho, permitindo obter uma perspectiva do que foi feito até agora nesta área, entre eles, Problem Frames e técnicas de identificação de aspectos.
- Capítulo 3. Abordagem de Problem Frames integrada com AORE. Apresentação da abordagem, em que é utilizado um exemplo simples para explicar o seu funcionamento.
- Capítulo 4. Caso de estudo. Este capítulo tem como finalidade aplicar a abordagem integrada de Problem Frames com AORE a um caso de estudo mais complexo (*Health-Watcher*), e comparando-a com outras abordagens.
- Capitulo 5. Conclusões. O último capítulo desta dissertação apresenta as principais contribuições da abordagem, bem como as suas limitações e sugere algumas direcções para trabalho futuro.
- Apêndice A: *problem frames* elementares.
- Apêndice B: Meta-modelo de Problem Frames.
- Apêndice C: Health Watcher.

2. Problem Frames e Engenharia de Requisitos Orientada a Aspectos

Este capítulo apresenta algumas técnicas e conceitos que serviram de base para o desenvolvimento da abordagem proposta nesta dissertação.

2.1. Problem Frames

Problem Frames [13] é uma abordagem de análise de problemas utilizada durante a fase de elicitação e especificação de requisitos. Esta identifica e caracteriza classes de problemas básicos que surgem durante o desenvolvimento do software. A utilização deste método promove a divisão de problemas maiores em subproblemas de menores dimensões, conferindo uma maior simplicidade ao seu estudo. A recomposição destes problemas mais pequenos é mais tarde necessária para que a solução do sistema inicial seja obtida.

O contexto principal deste método é realizar a descrição das interacções entre os domínios (*domains*) que pertencem e interagem com o sistema, existindo no mundo real do problema, onde cada domínio contém fenómenos de interfaces, designados por *phenomena*.

Na sua abordagem baseada em problemas, Jackson [13] definiu três tipos principais de diagramas: diagrama de contexto, diagrama de problema e diagrama de *problem frame*.

2.1.1. Diagrama de Contexto

Em [13], Jackson apresenta o diagrama de contexto (*context diagram*), como uma representação e contextualização do problema, onde se identificam todas as suas características e as interacções existentes entre as partes do mundo real que o caracterizam [8], separando-as em variados domínios, facilitando a sua posterior análise. Neste caso, o diagrama de contexto é definido por um domínio máquina (*machine domain*), que representa o sistema a ser desenvolvido e responsável pela solução, um ou mais domínios do problema em causa (*problem domains*) e as respectivas interfaces de fenómenos (modo como estão os domínios ligados) entre eles.

Os domínios do problema podem ser caracterizados por dois tipos distintos. O primeiro representa uma parte do mundo do problema que pode vir a ser estruturado e implementado (*designed domain*), de modo a representar algum tipo de informação. O segundo é um domínio onde as suas propriedades nos são fornecidas desde o início (*given domain*), não sendo possível alterá-las, podendo eventualmente ser apenas afectado o seu comportamento.

As interacções entre os diversos domínios existentes podem ser designadas por fenómenos partilhados (*shared phenomena*), tornando assim possível demonstrar o conjunto de eventos, estados e valores partilhados entre eles. A partir do momento em que o diagrama de contexto se encontra disponível, torna-se possível não só, situar o problema e todos os seus constituintes, bem como a forma como se encontram interligados, como é representado através da figura 2.1.

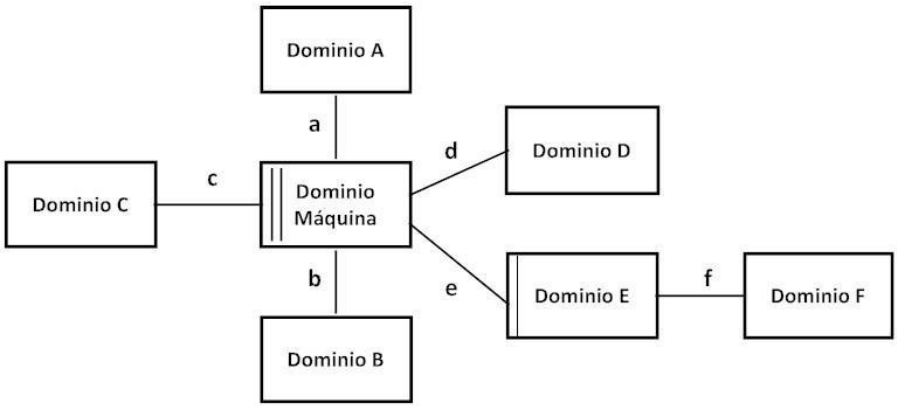


Figura 2.1: Diagrama de Contexto**.**

Como é observável na figura 2.1, os diversos domínios encontram-se ligados entre si, partilhando fenómenos, neste caso, exemplificado com recurso a letras. O domínio E (*designed domain*) é identificado pela risca, sendo que os restantes são domínios fornecidos (*given domains*). O domínio máquina é representado recorrendo a duas riscas.

2.1.2. Diagrama de problema

No entanto, de modo a permitir um ponto de partida para a análise cuidada e completa de um problema, é necessário definir um diagrama de problema (*problem diagram*), de forma a compreender o que efectivamente é o problema, como referido em [13]. Comparativamente ao diagrama de contexto, é apenas acrescentado um domínio, designado por requisito e

representado por um círculo a tracejado. Este corresponde às funcionalidades que o cliente pretende ver implementadas e que o sistema, através da máquina, terá de satisfazer [13, 8]. Um diagrama de problema demonstra explicitamente como este requisito se relaciona com os domínios existentes e quais os papéis que estes desenrolam no âmbito do problema. A figura 2.2 demonstra um exemplo padrão de um diagrama de problema.

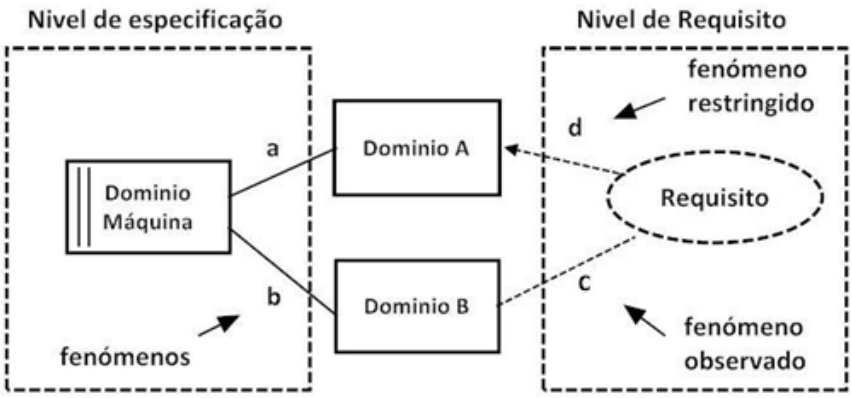


Figura 2.2: Diagrama de problema**.**

Observando a figura 2.2, podemos constatar que existem dois níveis distintos. O nível de especificação, como o nome indica, permite especificar as acções e os fenómenos partilhados entre os domínios. Já o nível de requisito é composto, para além do requisito, dos fenómenos observados e restringidos. O fenómeno restringido *"d"* é representativo do estado que se pretende alcançar e que se encontra descrito no requisito, necessitando, para isso, que alguns elementos sejam fornecidos à máquina. Já o fenómeno observado *"c"* representa, portanto, esses mesmos elementos, imprescindíveis para que *"d"* se possa verificar e esse mesmo requisito seja alcançado com sucesso.

A forma mais correcta de abordar o problema é, no entanto, decompô-lo num número finito de subproblemas, de menores dimensões e mais simples, permitindo assim uma maior facilidade na sua análise, compreensão e extensão, de maneira a que as suas soluções individuais contribuam para a solução do problema original.

Cada um destes novos subproblemas será considerado único e, como tal, terá direito ao seu próprio diagrama de problema, domínios e fenómenos, considerando sempre que todos os outros subproblemas estão solucionados, evitando, assim, a possibilidade de surgirem algumas confusões.

Estes subproblemas podem ser estruturados tendo em vista as suas principais características, onde cada um consistirá apenas numa projecção do problema original,

permitindo a existência de diversos domínios e informação partilhada em mais do que um subproblema.

2.1.3. Diagrama de problem frame e frame concern

Um *problem frame* pode ser definido como um padrão para a análise, estruturação e classificação de problemas de software já conhecidos. Normalmente este é descrito, de acordo com os seus requisitos, características dos domínios e fenómenos de interface.

Os vários tipos de domínios que podem surgir em cada situação podem ser classificados em três grupos:

- *Causal* – As suas propriedades incluem relacionamentos causais previsíveis, como por exemplo um sensor;
- *Biddable* – Caracterizado pela imprevisibilidade das suas acções, como o caso de uma pessoa;
- *Lexical* – Representação simbólica de dados, tal como um ficheiro de texto ou base de dados.

No que diz respeito às diversas classes de *problem frames* [13, 12], cada uma deve ser aplicada de acordo com o problema em causa. Existem cinco tipos de classe distintas:

- *Required behavior* – Aplicável a problemas onde é necessário construir uma máquina para controlar o comportamento de um domínio;
- *Commanded behavior* – Quando o comportamento de um domínio é afectado pelos comandos de um operador e é necessário construir uma máquina que aceite esses mesmos comandos e aplique o controlo;
- *Information display* – Casos em que se torna imprescindível obter do mundo real uma determinada informação e apresentá-la da forma desejada*;*
- *Simple workpieces* – Problemas onde é essencial criar ou editar um certo tipo de objecto léxico*;*
- *Transformation frame* – Situações em que se pretende transformar um tipo de dados, por intermédio de algum tipo de operação de processamento.

De modo a compreender as noções essenciais destas classes de problemas elementares, bem como a sua correcta forma de representação e especificação, vai ser utilizado o exemplo de um *Commanded Behaviour problem frame*, que será apresentado de forma mais detalhada. As restantes classes serão mencionadas no Apêndice A.

O diagrama de *problem frame Commanded behaviour* é caracterizado por um domínio, em que o seu comportamento é controlado por comandos a serem inseridos nalgum tipo de mecanismo, designado por domínio controlado (*controlled domain*). Contém um domínio operador (*biddable*), do qual são esperados certos comandos. É muito semelhante ao *Required Behaviour* com a diferença de que é adicionado o domínio operador.

O formato genérico para este tipo de problema é observável na figura 2.3, em que os fenómenos *MC!C1*, *DC!C2* e *OP!E4* representam as interacções entre os domínios *Máquina Controlo, Domínio Controlado* e *Operador*, e em que *C3* e *E4* simbolizam os estados objectivo e necessário, respectivamente, para que o requisito seja concretizado. O símbolo "!" é precedido pela abreviação do domínio que controla o fenómeno.

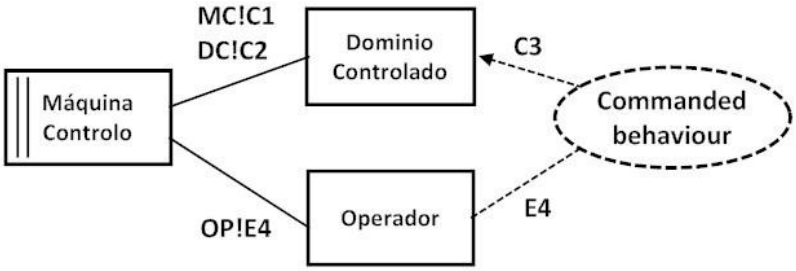


Figura 2.3: Problem frame *Commanded Behaviour***.**

Para que a caracterização do sistema seja realizada na sua totalidade, é indispensável complementar os diagramas com a utilização de descrições, que demonstrem que o problema está a ser correctamente compreendido e analisado. Isto permite identificar os passos para a sua solução.

É através de três tipos de descrição (descrição do domínio, especificação da máquina e requisito) que os argumentos apresentados ao cliente final serão suportados, de modo a convencer este último de que o problema se encontra a ser correctamente solucionado, embora, dando origem por vezes, a novos conflitos e *concerns*.

A explicação ao nível do requisito apresenta as diversas funcionalidades que terão de ser concretizadas, bem como o comportamento do sistema em resposta às mesmas. No caso da descrição do domínio, este realçará apenas o comportamento de cada domínio, quando origina ou é afectado por algum tipo de evento. No respeitante à especificação da máquina,

esta serve apenas para demonstrar a forma de agir e reagir da própria máquina, dependendo, no entanto, dos diferentes estados em que se encontra.

Estas descrições são elaboradas de modo a obter argumentos válidos e devem ser aplicadas a cada um dos *problem frames*, tendo em conta o problema em causa. Este conceito é designado por *frame concern* [13] (ver secção 3.2.4.5).

2.1.4. Meta-modelo de Problem frames

O conceito de *problem frames* foi proposto com intuito de compreender e caracterizar classes de determinados tipos de problema, principalmente no que diz respeito aos seus requisitos, domínios e comunicação entre eles (interfaces que consistem em fenómenos partilhados).

Em [11] é apresentado um meta-modelo formal de Problem Frames, expresso por intermédio de um diagrama de classes UML (*Unified Modeling Language*) [27] e uma notação de especificação formal OCL (*Object Constraint Language*) [11, 20], visando clarificar os seus diferentes constituintes e suas relações. O objectivo deste trabalho foi, facto considerado pelos autores, proporcionar uma Framework que servisse como um meio de análise sintáctica e validação semântica de *problem frames*, preparando a possibilidade de desenvolvimento de uma ferramenta para o suporte desta abordagem.

O meta-modelo encontra-se maioritariamente representado na figura 2.4. Outras figuras que complementam este modelo estão presentes no Apêndice B.

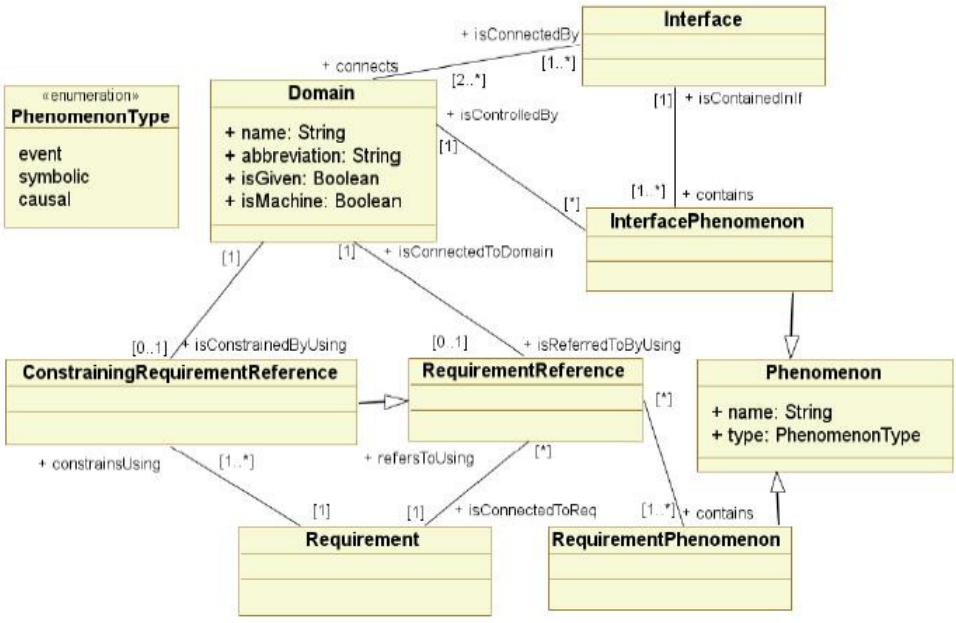


Figura 2.4: Meta-modelo de Problem Frames [21].

De acordo com Jackson [13] e [11], um *problem frame* é composto por domínios, um requisito, interfaces e referências de requisito.

O modelo da figura 2.4 demonstra as relações existentes entre os elementos de Problem Frames. No diagrama de classes de [11], as interfaces ligam domínios e contém fenómenos. Cada domínio é conectado por pelo menos uma interface e um fenómeno está contido em exactamente uma interface, portanto um fenómeno só pode ser controlado por esse mesmo domínio. Cada fenómeno pode ser designado por fenómeno de interface, caso esteja contido numa, ou fenómeno de requisito, isto se estiver contido nas referências de requisitos.

Um requisito de um *problem frame*, refere-se a vários domínios através das suas referências, mas pode no entanto restringir pelo menos um domínio. Um domínio pode ser referenciado ou restringido por um requisito. Cada referência do requisito, seja esta restringida ou não, contém sempre um ou mais fenómenos.

2.2. Engenharia de Requisitos Orientada a Aspectos

A Engenharia de Requisitos Orientada a Aspectos (EROA) é parte do movimento *Early Aspects* (http://early-aspects.net) e está orientado sobretudo para a identificação, modularização, representação e composição de assuntos transversais (*crosscutting concerns*) ao nível dos requisitos, sejam estes funcionais ou não funcionais, nos estágios iniciais do desenvolvimento de software.

Estes assuntos são requisitos ou características comportamentais que se pretendem ver considerados no sistema em desenvolvimento e que têm um efeito observável noutras funcionalidades. A sua ênfase é dada à composição, pois após modularizar um aspecto é necessário especificar como ele influencia, afecta ou restringe os requisitos com que se encontra relacionado.

Um aspecto pode ser modelado por intermédio de uma abordagem de análise de requisitos que consiga encapsular, separar e tratar cada *crosscutting concern* dos restantes.

Os aspectos podem ser considerados em qualquer mecanismo de separação de *concerns* tais como *viewpoints*, *use cases*, *goals* ou *problem frames*. As abordagens orientadas a aspectos tem o seu foco em descrever como um aspecto pode restringir ou influenciar determinados requisitos de um sistema.

Estas podem ser caracterizadas, essencialmente, pela existência de meios ou ferramentas, para a identificação de propriedades transversais e a capacidade de as modularizar durante a especificação de requisitos.

Nesta dissertação, vai ser dado destaque à abordagem AORE [22], que combina uma técnica orientada a *viewpoints* com aspectos, sendo esta descrita de seguida.

2.2.1. AORE

Esta técnica de análise orientada a aspectos, apresentada em [22], tem como objectivo demonstrar como modularizar e especificar requisitos aspectuais, requisitos não aspectuais e regras de composição. A abordagem AORE propõe um modelo de separação de requisitos não funcionais aspectuais. Esta é baseada em PREView [9, 26, 27] e XML (*eXtensible Markup Language*) [30].

PREView é um método de engenharia de requisitos orientada a *viewpoints*, onde cada *viewpoint* é considerado uma entidade que contém alguma, mas não toda, informação sobre o problema a ser resolvido ou o sistema a desenvolver. Existem dois tipos de requisitos. Os funcionais, que são considerados específicos de cada *viewpoint* e organizados relativamente ao ponto de vista próprio. E aqueles que são globais (*concerns*) a todo o sistema. Em PREView, a um *concern* corresponde um objectivo não funcional do sistema. É por esta razão que muitos destes *concerns* se podem encontrar presentes em mais do que um requisito. De notar a diferença conceitual de um *concern* em PREView (sempre não funcional) e um *concern* em EROA (funcional ou não-funcional).

AORE procura incidir principalmente na modularização e composição de *concerns* que afectem e restrinjam outros requisitos. Também proporciona uma forma de identificar, mas também de solucionar os diversos tipos de conflitos que possam eventualmente surgir, utilizando esses aspectos em fases mais avançadas do desenvolvimento.

Esta abordagem possui uma ferramenta de suporte designada por ARCaDe (*Aspectual Requirements Composition and Decision*) [22], que permite manipular requisitos que se encontram organizados em viepoints e *concerns* descritos em XML.

O modelo da Figura 2.5 representa os passos a seguir para lidar com aspectos usando a abordagem AORE [22].

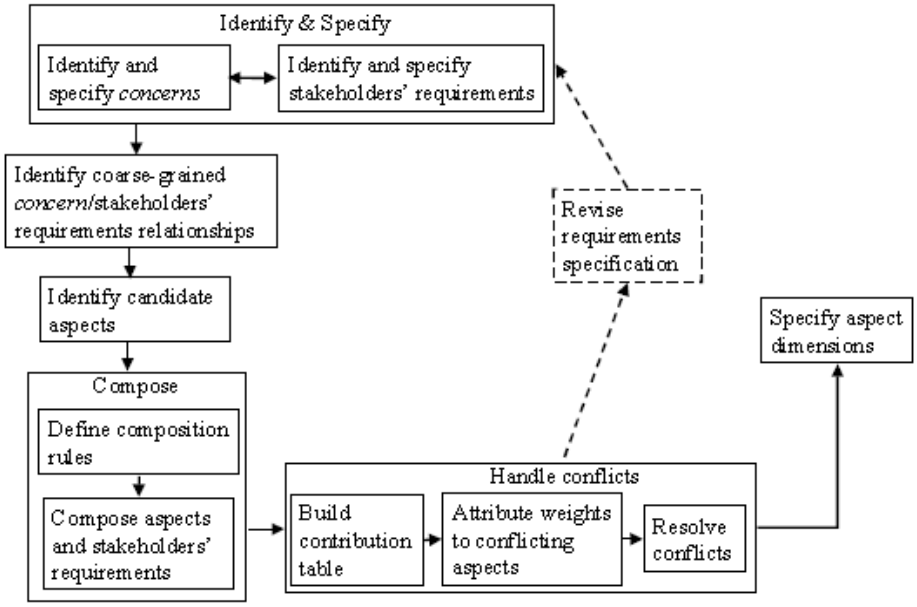


Figura 2.5: Modelo AORE [22].

Primeiramente, é necessário identificar e especificar requisitos e *concerns*, para que seja possível efectuar um relacionamento entre eles. A abordagem utiliza, para este efeito, uma matriz, permitindo assim determinar todos os *concerns* que influenciem mais do que um requisito de diferentes *viewpoints*, passando a designar-se por aspectos candidatos. A tabela 2.1 demonstra a relação entre cada *concern* e *viewpoints* (SRi). Na tabela o $Concern_2$ e $Concern_n$ são *crosscutting concerns* pois afectam mais de um *viewpoint*.

Tabela 2.1: Relacionamento entre *concerns* e requisitos [22].

SR / ***Concerns***	SR_1	SR_2	…	SR_n
$Concern_1$				✓
$Concern_2$		✓		✓
…				
$Concern_n$	✓	✓		

Posteriormente, é realizada a definição de regras de composição, a cada *crosscutting concern* identificado, que irão ajudar a compreender e a especificar o modo, como um requisito aspectual influencia o comportamento de um conjunto de requisitos não aspectuais nos diversos *viewpoints*.

Após este processo de reconhecimento de aspectos e respectiva definição de todas as regras de composição, inicia-se então a identificação e resolução de conflitos, por intermédio de uma matriz em que se relacionam os diversos aspectos, onde um poderá contribuir

positivamente (+) ou negativamente (-) para qualquer um dos outros. A tabela 2.2 ilustra a matriz de identificação de conflitos.

Tabela 2.2: Relacionamento entre aspectos [22].

Aspects / Aspects	$Aspect_1$	$Aspect_2$	…	$Aspect_n$
$Aspect_1$		+		
$Aspect_2$				–
…				
$Aspect_n$				

Se dois aspectos em conflito (contribuam negativamente entre si) afectam um mesmo *viewpoint,* são lhes atribuídos pesos (valores no intervalo [0..1]) com o fim de estabelecer a prioridade que o *viewpoint* dá em relação a cada aspecto com o fim de resolver o conflito.

Enquanto esta resolução de conflitos ocorrer, torna-se então essencial rever todas as especificações dos requisitos, sejam eles funcionais ou aspectuais e até mesmo as regras de composição, dando origem a um novo ciclo.

2.2.2. Outras abordagens

Nesta secção serão apresentadas outras abordagens orientadas a aspectos.

2.2.2.1. Cenários com aspectos

Uma outra forma de representar aspectos durante a análise de requisitos em engenharia de software é através da utilização da modelação baseada em cenários. Em [3, 28], utilizando *use cases* refinados em cenários, foi proposto um modo de compor cenários aspectuais e não aspectuais, para que estes possam ser simulados como um todo. Um cenário pode ser visto como uma forma de observar o comportamento de um sistema. Neste caso, os cenários não aspectuais são representados por diagramas de sequência, enquanto os cenários aspectuais (cenários presentes em mais do que um cenário) são modelados por intermédio de *Interaction Pattern Specifications (*IPSs*).* Um IPS define um padrão de interacções entre os seus participantes, consistindo nas trocas de um determinado número de mensagens durante a sua linha de vida. Cada elemento aspectual presente num IPS desempenha um papel (*role*).

Sempre que sejam verificadas certas condições, esse elemento é instanciado e vai corresponder a elementos concretos.

A abordagem inicia-se pela determinação dos requisitos funcionais e não funcionais exigidos pelo sistema, sendo que os primeiros devem ser descritos por *use cases*. Após uma análise do relacionamento entre os dois tipos de requisitos, é possível detectar aqueles que sejam transversais, determinando, assim, os aspectos a ter em conta no desenvolvimento do software. Desta forma, tanto os aspectos, bem como as funcionalidades do sistema, são representados por IPSs e diagramas de sequência (cenários) respectivamente.

A figura 2.6 procura exemplificar este processo.

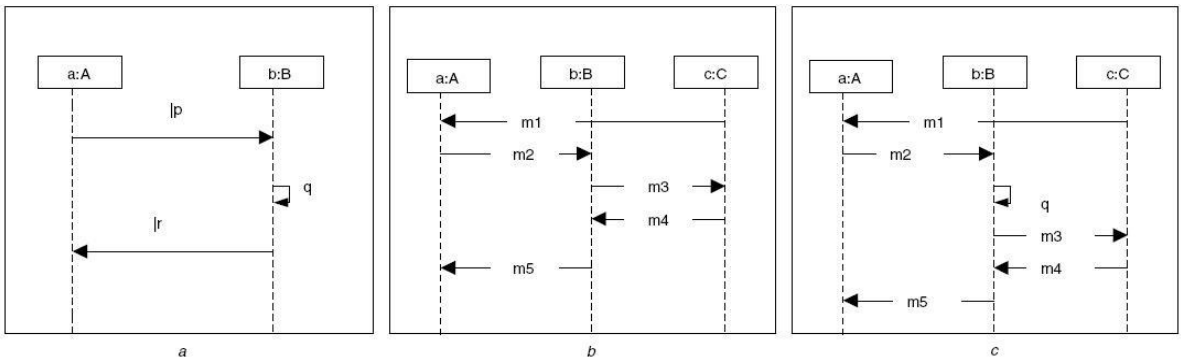


Figura 2.6: IPSs [28].

Para que a junção e composição entre os cenários seja efectuada com sucesso, tal como é demonstrado na figura 2.6 são necessários três elementos: o IPS (a), o diagrama de sequência (b) e o operador de integração (IN, OR ou AND), que define o modo como o primeiro deve ser integrado com o segundo. Neste caso, analisando o IPS, sempre que ocorra o envio de uma qualquer mensagem (|*p*) de *a:A* para *b:B* e um posterior envio de outra mensagem (|*r*) no sentido de inverso, a execução de *q* em *b:B* deve ocorrer. No diagrama de sequência, observa-se que *m2* e *m5* respeitam as condições do IPS. Portanto, no diagrama final (c), através do operador IN inclui-se o acontecimento *q*, através de *pattern matching*.

Existem três operadores diferentes, OR, AND e IN, tal como apresentado em [28]. O primeiro especifica que o IPS e o cenário não aspectual são alternativos, sendo que numa determinada altura existirá uma escolha entre eles, onde o escolhido será aquele a ser executado. O operador AND define que a execução será feita de forma concorrente. Por último, o operador IN insere a execução do cenário aspectual no diagrama de sequência de modo sequencial.

A utilização de *pattern matching* para instanciação de elementos é de extrema importância para o trabalho desta dissertação, como será demonstrado no capítulo 3.

2.2.2.2. VAODA

Uma outra abordagem que utiliza *viewpoints* para detecção de aspectos é designada por VAODA (*Viewpoint and Aspect Oriented Domain Analysis Approach*) e é apresentada em [23]. Esta tem o seu foco na análise de domínios com aspectos e utiliza, essencialmente, *viewpoints* para especificar e estruturar os requisitos funcionais dos domínios. Para tal, o modelo de PREView [26] é estendido com aspectos. É introduzido também o conceito de *viewpoint* aspectual, composto exclusivamente por requisitos semelhantes, mas que sejam transversais a mais do que um *viewpoint*. Um dos passos do seu processo é compor cada *viewpoint* aspectual com os *viewpoints* originais, por intermédio da instanciação das partes que compõem os requisitos *crosscutting* (*roles*). O processo de instanciação é realizado através de regras de composição intuitivas.

2.2.2.3. MATA

MATA (*Modeling Aspects using a Transformation Approach*), é uma ferramenta de modelação UML (*Unified Modeling Language*), baseada em transformações de grafos, com o intuito de especificar e compor aspectos. Em [29] esta abordagem foca principalmente diagramas de classes, sequência e estado. MATA considera a composição de aspectos como um caso especial de transformação de grafos através da aplicação de certas regras, utilizando para tal, uma ferramenta designada por AGG [29].

A figura 2.7 demonstra a aplicação de uma regra MATA [29].

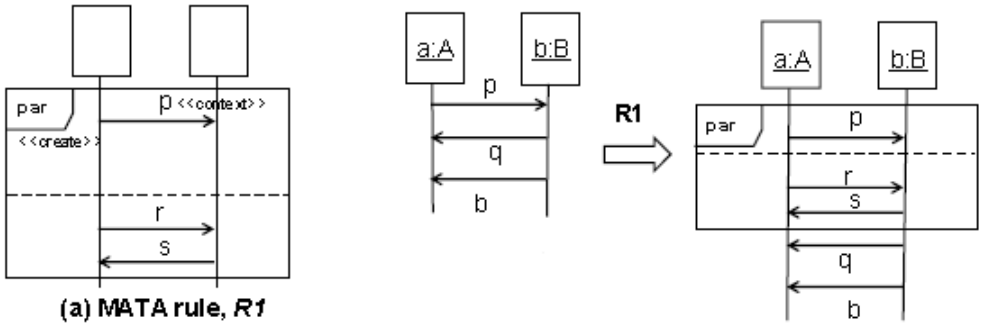


Figura 2.7: Aplicação de regra MATA [29].

Neste exemplo simples, é possível verificar a aplicação da regra no diagrama de sequência base, em que o cenário aspectual (a) especifica o comportamento que deve ser introduzido no diagrama de sequência base e quando este deve ser inserido. A regra *R1* deve, portanto, ser aplicada sempre que uma mensagem *p* ocorra no diagrama base, introduzindo também *r* e *s*, onde o fragmento ***par*** consiste numa execução paralela de todas elas.

2.2.2.4. AORA

Uma outra abordagem em engenharia de requisitos que visa a identificação e modularização de todos os tipos de *concerns* (funcionais ou não funcionais), considerando também os *crosscutting concerns* é designada por AORA (*Aspect-oriented Requirements Analysis*) [4]. Esta permite a definição de uma linguagem que especifica e compõe *concerns* ou a resolução de conflitos, deixando em aberto a utilização dos *concerns* identificados, sejam eles ou não *crosscutting*, em fases mais avançadas do desenvolvimento.

O modelo apresentado em [4], tenta providenciar mecanismos que possibilitem identificar e modularizar todos os *concerns* existentes.

Após a identificação, decomposição e especificação de cada *concern*, o passo mais relevante é a sua composição. Inicialmente, este processo é realizado através da determinação de *match-points* relativamente a requisitos funcionais, possibilitando deste modo identificar os *crosscutting concerns* existentes [5]. Por último, são aplicadas as regras de composição, a fim de determinar as prioridades de cada *concern* no âmbito geral da aplicação. A gramática utilizada para este efeito é a apresentada na figura 2.8.

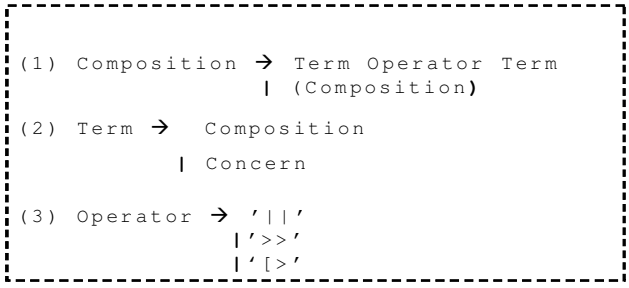


Figura 2.8: Gramática para regras de composição de *concerns*.

No caso do aparecimento de conflitos que necessitem de atenção e que devem ser solucionados, é utilizado um processo para encontrar a melhor alternativa entre as várias possíveis (*Multicriteria Decision Making*), por intermédio de uma técnica que garanta a

consistência lógica das decisões (*Analytical Hirarquical Process*), através da expressão de preferências entre os critérios, ou atribuindo pesos a cada opção.

2.2.2.5. Vision e use cases

Vision [1, 2] é um método de elicitação de requisitos orientado a *viewpoints*, baseado em PREView [9, 26, 27] e VORD (*Viewpoint-Oriented Requirements Definition*) [15]. O processo de VORD inclui a identificação, descrição e análise de *viewpoints* para descobrir inconsistências e conflitos. O que distingue Vision de outras abordagens é a sua capacidade para descrever associações e agregações entre *viewpoints*, incorporando modelos UML. No entanto, em [2], é apresentada uma adaptação de Vision que procurou igualmente integrar a detecção de aspectos (funcionais e não funcionais) na sua abordagem.

2.3. Problem Frames e aspectos

Nesta secção serão apresentadas algumas abordagens que integram aspectos em Problem Frames.

2.3.1. Problem Frames e aspectos

Uma tentativa de integração de aspectos em abordagens de engenharia de requisitos foi introduzida em [18], permitindo demonstrar como um modelo de Problem Frames pode ser refinado de forma a determinar o modo de identificar e modularizar aspectos (ver figura 2.9).

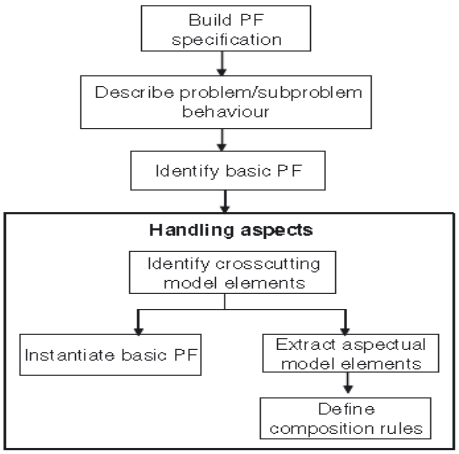


Figura 2.9: Modelo de identificação de elementos *crosscutting* através de *problem frames* [18].

Esta abordagem teve como objectivo apresentar um processo, descrevendo os diversos passos para a identificação de elementos pertencentes ao modelo de *problem frames*, que sejam *crosscutting* (presentes em diversos elementos do modelo). Desta feita o processo descrito na figura 2.9 tem o seu principal foco no reconhecimento de requisitos aspectuais (presentes em diversos diagramas de problema) e *problem frames* aspectuais (identificado em mais que um subproblema) de um problema.

O processo ilustrado pode ser caracterizado mais detalhadamente pelos seguintes pontos:

- Providenciar um ponto de partida para a análise de um problema, construindo o diagrama de contexto, de modo a determinar onde se localizam os problemas, identificar os requisitos e principais *concerns*.
- Descrever o problema, subdividindo-o em subproblemas mais simples e reduzidos, considerando os fenómenos partilhados entre os vários domínios e descrevendo o seu comportamento.
- Dar seguimento à análise dos subproblemas até que estes se possam enquadrar num *problem frame* elementar, como definido em [13].
- Identificar e especificar requisitos e *problem frames* aspectuais, determinando os problemas onde o requisito está envolvido e definindo regras de composição para capturar a forma como um aspecto os afecta.

É apresentado também um meta-modelo, como apresentado na figura 2.9. Este demonstra as ligações entre os vários elementos de Problem Frames, com extensão para aspectos. Neste, é possível observar-se as características *crosscutting* dos *problem frames,* quer seja com *concerns*, fenómenos, domínios, problemas ou até mesmo requisitos. Isto significa que qualquer um dos elementos pertencentes ao modelo de Problem Frames, pode ser candidato a aspecto, caso se encontre espalhado pelas várias partes do problema em causa. Esta é ainda uma abordagem inicial, onde a identificação de aspectos e sua composição é ainda limitada, facto reconhecido pelos autores, pois não contempla a detecção de requisitos não funcionais aspectuais.

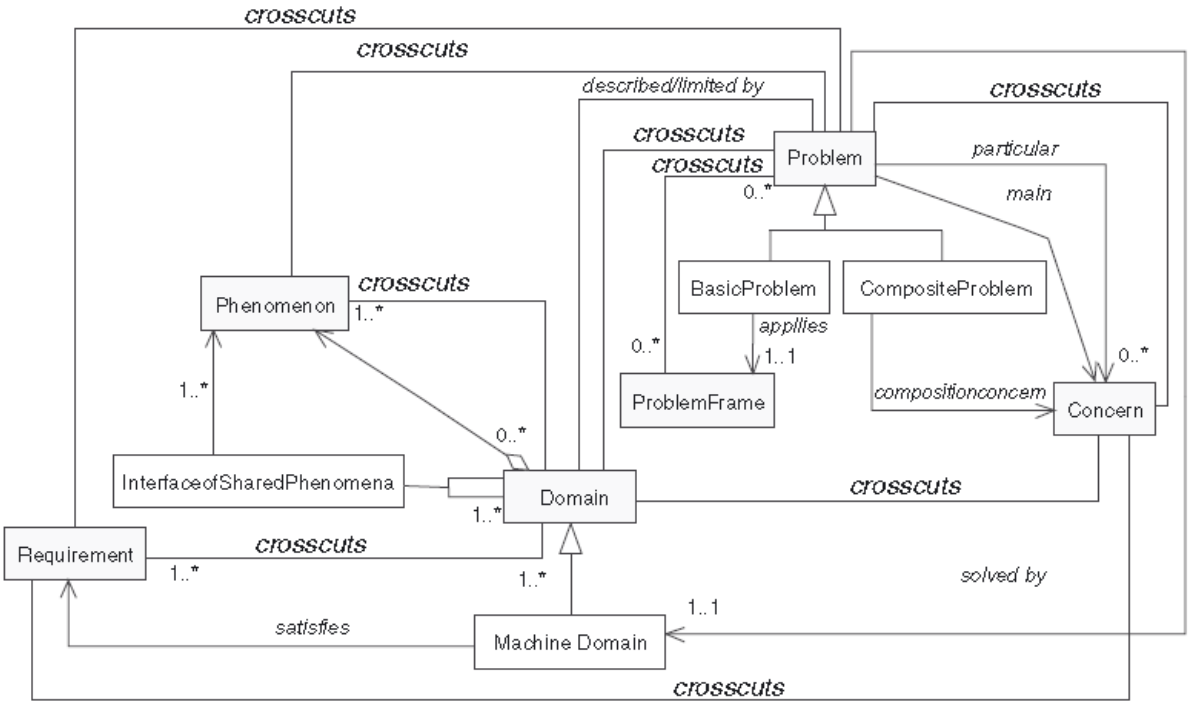


Figura 2.10: Meta-modelo de Problem Frames orientado a aspectos [18].

O modelo apresentado na figura 2.10 vai permitir atingir um dos objectivos propostos nesta dissertação, isto é, desenvolver um meta-modelo que integre Problem Frames e a técnica AORE. As classes Problem, BasicProblem e CompositeProblem serão adaptadas ao meta-modelo integrado de Problem Frames com AORE (ver secção 3.3.2).

2.3.2. Problem Frames e cenários aspectuais

Numa outra abordagem [17], foi apresentado um modo diferente de representar, identificar, especificar, modularizar e compor aspectos em Problem Frames (PF), como se pode verificar na figura 2.11.

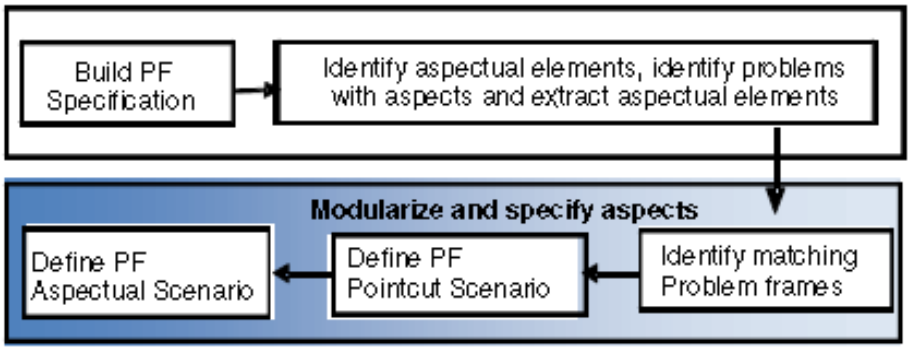


Figura 2.11: Processo de identificação e especificação de aspectos através de *problem* f*rames* [17].

Para que tal seja possível, é necessário primeiramente especificar todos os *problem frames* tal como descrito por Michael Jackson em [13]. Posteriormente é efectuada a identificação de todos os elementos aspectuais, através da análise de relações entre subproblemas e os *problem frames* existentes como demonstrado em [18].

Para modularizar e especificar cada requisito aspectual utilizam-se cenários tal como em [3], identificando se existe algum *problem frame* que corresponda ao requisito aspectual, definindo-se o *PF Poincut Scenario* (cenário padrão que deve ser considerado para fazer corresponder o aspecto, através da sua representação por intermédio de um diagrama problema ou *problem frame*).

De modo a definir as regras de composição que determinam a forma como um aspecto afecta cada elemento do modelo onde este exista, é descrito o *PF aspectual scenario*. Este é representado por um diagrama híbrido (contém um diagrama de problema associado a um *problem frame*), conjuntamente com um diagrama de sequência representativo da ordem de execução de cada comportamento dos diversos elementos do sistema, onde podem ser também incluídos ou removidos domínios e fenómenos.

Após se ter encontrado o formato genérico para resolver cada requisito aspectual, é indispensável efectuar a sua composição integrando o modelo base e os seus domínios e fenómenos concretos.

2.3.3. Requisitos de segurança em Problem Frames

Uma tentativa de demonstrar como a representação de *threats* (ameaças que podem prejudicar o contexto de um problema) em *crosscutting concerns*, pode auxiliar na determinação dos efeitos de requisitos não funcionais, nomeadamente aqueles baseados em segurança (confidencialidade, integridade, e disponibilidade), sobre diversas funcionalidades, é ilustrada em [10]. Esta abordagem utiliza *problem frames*, como ferramenta de organização de informação, para que a derivação de requisitos de segurança seja facilitada. Para isso, são utilizados conceitos orientados a aspectos, já que normalmente, este tipo de requisitos tem um impacto em várias funcionalidades do sistema.

Neste caso, cada *problem frame* desempenha a função de ilustrar requisitos funcionais e as fraquezas do sistema. Estas potenciais vulnerabilidades podem ser colmatadas com a imposição de restrições de segurança, facilitando o processo de especificação.

Desta feita, a descrição de ameaças é efectuada por intermédio da sua representação na forma de tuplos, em que a composição com requisitos funcionais, a fim de derivar novos atributos de segurança, é realizada por meio de um processo iterativo. Este processo iterativo, tal como apresentado em [10] contém 4 passos:

- Identificação de objectos (*assets*) no contexto do problema.
- Identificação de ameaças (*threats*) nos objectos realizando a sua descrição.
- Determinação de relacionamentos *crosscutting* entre os requisitos funcionais e as ameaças, obtendo o subproblema em que os objectos se encontram. Assim, permite-se a enumeração de restrições que enfraqueçam as vulnerabilidades, bem como a estruturação do conjunto de tuplos respectivo, representado por *{(threat, asset, subproblem)}*.
- Identificação de conflitos entre as restrições obtidas do ponto anterior.

Este processo ocorre até que não surjam mais requisitos para derivação, já que esta abordagem é limitada exclusivamente aos aspectos de segurança.

2.4. Resumo

Neste capítulo foi apresentada a abordagem de Problem Frames [13] e suas principais características, nomeadamente, os passos a seguir para analisar e contextualizar um problema. Desde a composição de um diagrama contexto, passando pelo reconhecimento dos requisitos e consequente decomposição dos diagramas de problemas, até à correspondência com os *problem frames* básicos. Também de mencionar, o estudo realizado sobre o meta-modelo apresentado em [11].

No que diz respeito aos conceitos principais de EROA, foram referidas várias abordagens que contemplam a identificação e modularização de aspectos, de entre elas, a abordagem AORE [22] que será utilizada para integrar com a abordagem de Problem Frames.

Ainda neste capítulo foram igualmente descritas três abordagens existentes que integram já conceitos aspectuais no modelo de Problem Frames. Desta feita, foi possível verificar o que até ao momento foi desenvolvido nesta área, realçando a importância para o método e meta-modelo apresentado na secção 2.3.1.

No próximo capítulo será então apresentada e demonstrada por intermédio de um exemplo, a abordagem proposta nesta tese.

3. Abordagem de Problem Frames integrada com AORE

A existência de *crosscutting concerns* durante o desenvolvimento de software pode comprometer a qualidade de implementação de um sistema, uma vez, que este pode conter elementos transversais a vários módulos, que por seu lado, poderiam ser melhor modularizados. Daí que se torne essencial tomar providências para minimizar os seus efeitos, logo nas fases iniciais do desenvolvimento do software, mais concretamente ao nível dos requisitos.

A abordagem desenvolvida nesta dissertação baseia-se na técnica de elicitação de requisitos AORE [22] combinada com Problem Frames [13]. O objectivo da integração destas duas abordagens é possibilitar a contextualização, de um modo mais completo, do problema que se encontra sobre estudo. Conforme apresentado no capítulo 2.2.1, a abordagem AORE integra os conceitos de *viewpoints* e aspectos. A importância da utilização de *viewpoints* expressa-se, essencialmente, pela necessidade de obter mais informação acerca de um problema de software, considerando as várias perspectivas do mesmo. Estes visam facilitar a descoberta, de uma forma mais completa e estruturada, dos requisitos funcionais e o seu respectivo estudo. O uso de aspectos, por sua vez, dá a possibilidade de representar e descrever cada aspecto, quer seja ele funcional ou não.

Por fim, utilizando o modelo de Problem Frames, vai permitir, através da operacionalização de cada aspecto não funcional, realizar a sua representação. Deste modo, o processo de integração proposto possibilita fazer uso de cada *viewpoint* para determinar concretamente os domínios envolvidos no problema, assegurando um método mais aperfeiçoado de elicitação e análise de requisitos.

3.1. Modelo de Problem Frames integrado com aspectos

O processo de elicitação e especificação de problemas proposto, procura utilizar as características fundamentais da abordagem de Problem Frames para descrever cada problema que possa surgir na análise de um sistema de software, tendo em conta a necessidade de lidar

com requisitos aspectuais (funcionais ou não funcionais). A técnica AORE será utilizada para a identificação concreta dos aspectos.

A abordagem aqui proposta é composta por cinco principais actividades:

- Identificar e especificar *viewpoints* e *concerns*;
- Identificar aspectos não funcionais;
- Identificar aspectos funcionais;
- Utilizar Problem Frames para representar problemas e requisitos aspectuais;
- Aplicar regras de composição a cada aspecto.

O modelo ilustrado na figura 3.1 demonstra o processo de utilização de métodos de identificação e modularização de aspectos, integrando Problem Frames.

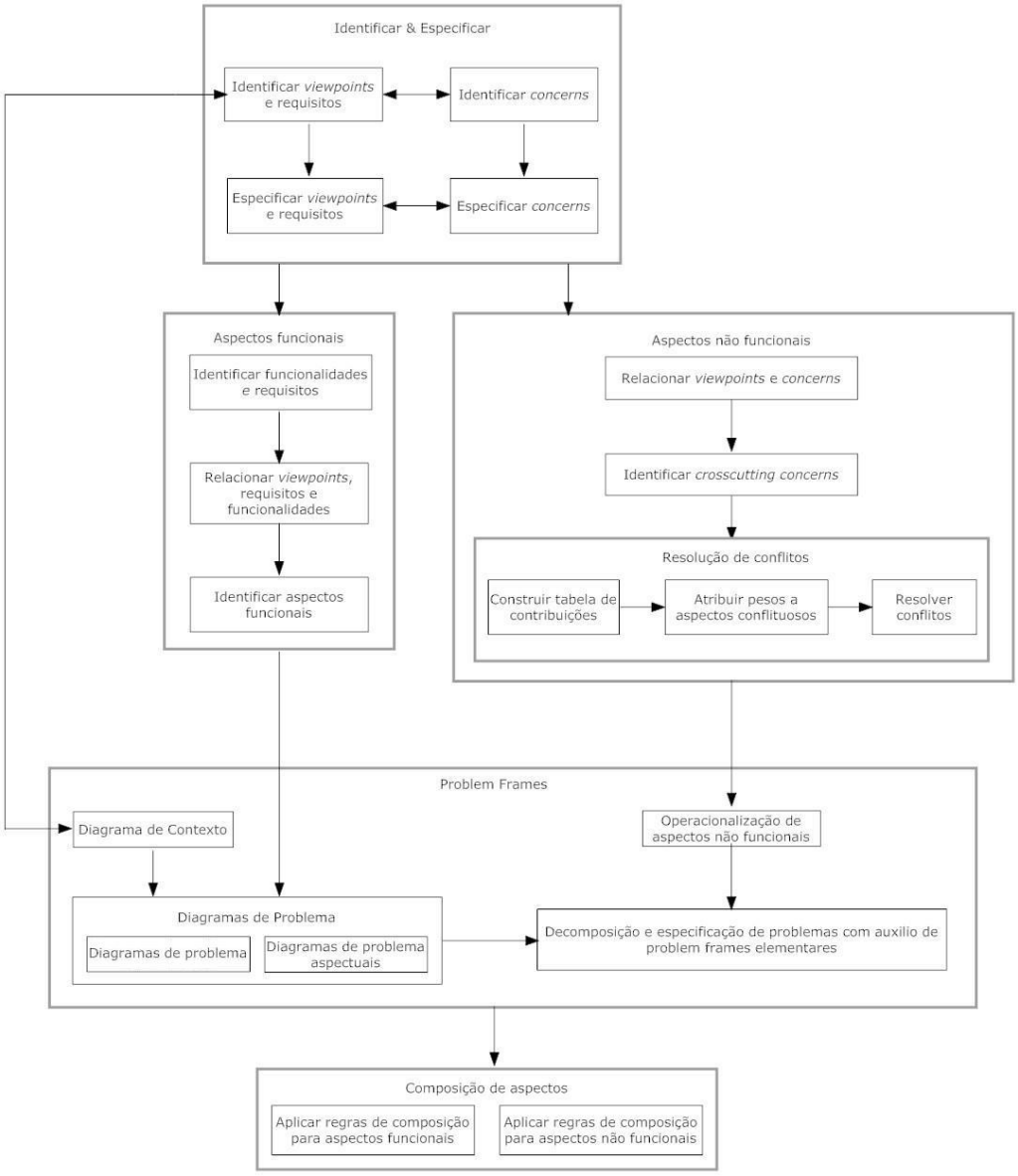


Figura 3.1: Processo integrado de PF e AORE**.**

Previamente, é indispensável determinar quais as funcionalidades pretendidas para o sistema. Este procedimento pode ser realizado através de entrevistas com os clientes, observação de tarefas, entre outras.

A partir deste ponto é preciso identificar quais os elementos relevantes para o funcionamento desejado do sistema, iniciando-se assim a utilização da abordagem proposta na figura 3.1.

Este processo começa com a análise do problema, possibilitando assim a identificação de cada *viewpoint* e especificação dos seus requisitos funcionais. À medida que cada *viewpoint* é identificado são também reconhecidos alguns dos *concerns* (requisitos não funcionais) pretendidos para o sistema. De tal modo, que possibilita identificar e descrever os *concerns* de uma forma conjunta com os *viewpoints*.

A descrição de cada *viewpoint* baseia-se nos atributos existentes no template do modelo apresentado em [1, 22], ver tabela 3.1, e que foram considerados úteis para a implementação desta abordagem.

Tabela 3.1: Template para um *viewpoint***.**

Atributos do Viewpoint	Descrição do Atributo
Nome:	Utilizado para identificar e deve ser seleccionado de forma a reflectir o foco do *viewpoint.*
Foco:	Define o viewpoint tendo em conta a sua perspectiva.
Concerns:	Por defeito contém todos os *concerns* existentes no sistema, no entanto, alguns podem ser eliminados se ficar provado que não tem qualquer papel para com o *viewpoint*.
Fontes:	As fontes do *viewpoint* são identificações explícitas das fontes da informação associadas com o *viewpoint*.
Sub-VP:	Conjunto de sub-*viewpoints* que contêm requisitos complementares ao *viewpoint* principal. Estes requisitos são exclusivos do sub-*viewpoint* em causa.
Requisitos:	Este é o conjunto de requisitos encontrados através da análise do sistema pela perspectiva do *viewpoint* e pela consulta com as fontes.

Também a descrição de cada concern é realizada e baseada nos atributos considerados relevantes para esta abordagem, presentes no template do modelo apresentado em [1, 22], ver tabela 3.2.

Tabela 3.2: Template para um concern**.**

Atributos do *Concern*	**Descrição do Atributo**
Nome:	Representa o nome do *concern*.
Descrição:	Utilizado para identificar e descrever a necessidade do requisito não funcional.
Viewpoints influenciados:	Conjunto de *viewpoints* afectados pelo concern.
Requisitos:	Conjunto de requisitos que descreve o *concern*.

A elaboração do Diagrama de Contexto, no âmbito de Problem Frames, pode ocorrer logo na fase inicial do processo integrado, como descrito na figura 3.1, fazendo a correspondência dos *viewpoints* identificados até ao momento, com os domínios do problema.

A realização de um cruzamento entre os *concerns* (aqui corresponde a propriedades não funcionais) e *viewpoints* obtidos permite determinar os elementos aspectuais envolvidos na análise de software, e assim identificar os *crosscutting concerns*. A tabela 3.3 ilustra *concerns* transversais a mais de um *viewpoint* (expresso por VP).

Tabela 3.3: Cruzamento entre *concerns* e *viewpoints***.**

VP / Concerns	VP_1	VP_2	…	VP_n
C_1		X		
C_2	X	X		
…				
C_n	X			X

Após todos os aspectos não funcionais terem sido reconhecidos (C_2 e C_n afectam mais que um *viewpoint*), resta procurar resolver as situações conflituosas entre eles, através da construção de uma matriz de contribuições e atribuição de pesos a cada aspecto conflituoso. Este processo é descrito com maior detalhe mais à frente neste capítulo, na secção 3.2.2.3.

Os aspectos funcionais são obtidos por intermédio do cruzamento entre as funcionalidades desejadas para o sistema e os requisitos de cada *viewpoint* que as compõem. A matriz da tabela 3.4 demonstra como se realizam estes relacionamentos.

Tabela 3.4: Matriz de relacionamento entre *viewpoints*, requisitos e funcionalidades**.**

Funcionalidades / Viewpoints	**Funcionalidade$_1$**	**Funcionalidade$_2$**
	Requisitos de *Viewpoints*	
VP$_1$	VP$_1$.R1	VP$_1$.R1
VP$_2$	VP$_2$.R2; VP$_2$.R4;	VP$_2$.R2;
…		
VP$_n$	VP$_n$.R1;	VP$_n$.R5;

Observando a tabela 3.4, os requisitos VP1.R1 e VP2.R2 são considerados aspectuais, pois são transversais a ambas as funcionalidades, devendo ser separados e removidos desta mesma matriz (ver com maior detalhe na secção 3.2.3). As tabelas 3.5 e 3.6 demonstram a modularização dos aspectos funcionais identificados e a sua remoção da matriz de relacionamento acima apresentada, respectivamente.

Tabela 3.5: Aspectos funcionais**.**

Aspecto funcional	**Requisitos**
AspVP1.R1	VP$_1$.R1 – Requisito 1 de *viewpoint* 1
AspVP2.R2	VP$_2$.R2 – Requisito 2 de *viewpoint* 2

Tabela 3.6: Matriz de relacionamento entre *viewpoints*, requisitos e funcionalidades com remoção de aspectos funcionais**.**

Funcionalidades / Viewpoints	**Funcionalidade$_1$**	**Funcionalidade$_2$**
	Requisitos de *Viewpoints*	
VP$_1$		
VP$_2$	VP$_2$.R4;	
…		
VP$_n$	VP$_n$.R1;	VP$_n$.R5;

Após o tratamento de todos os aspectos, estes já podem ser representados no contexto de Problem Frames.

Inicialmente, surge a necessidade de completar o Diagrama de Contexto para ter uma melhor visão e compreensão do problema em causa. Cada funcionalidade e aspecto (funcional ou não funcional) devem ser representados através de diagramas de problema. No caso dos aspectos funcionais, a sua representação em forma de diagramas aspectuais recorre à utilização de domínios e fenómenos genéricos (ver secção 3.2.4.4). Em termos de aspectos não funcionais, é necessário previamente realizar a sua operacionalização (ver secção 3.2.4.5). Todos os diagramas devem então ser alvo do processo de decomposição e especificação com

o auxílio de *problem frames* elementares, tal como apresentado em [13], de forma a garantir uma mais correcta análise ao problema.

O passo final da abordagem visa aplicar regras de composição entre todos os problemas aspectuais com os problemas iniciais. Estas regras utilizam principalmente a composição através de parte gráfica com a utilização de *pattern matching* (ver secção 3.2.5).

3.2. Explicando a abordagem através de um exemplo

Para ilustrar a aplicabilidade desta abordagem, vai ser tomado como exemplo o seguinte problema de controlo de acesso a parques de estacionamento, onde é necessário desenvolver um sistema que utiliza identificadores do tipo via verde:

"Os identificadores obtêm-se através de um simples processo de adesão, onde o cliente fornece os seus dados pessoais, do seu cartão de débito e ainda do veículo a registar. É necessário fazer a activação do identificador numa caixa de multibanco (ATM) associando este ao cartão de débito. Para aceder a um parque basta que o identificador, colado no pára-brisas do veículo, aproxime-se de uma das cancelas especiais que se abrirá se ele for válido e uma luz verde acenderá. Para sair do parque o procedimento é semelhante. A quantia a pagar depende do tempo gasto no parque e é mostrada num display acoplado a uma cancela de saída. Ainda, qualquer reclamação relacionada ao sistema (e.g. erro na leitura do identificador na entrada ou na saída, ou no cálculo da quantia a ser debitada) pode ser feita a um funcionário do parque que deverá registá-la no sistema. O valor a pagar é enviado semanalmente pelo sistema ao banco onde o cliente tem a conta para os débitos. Se houver algum problema na transacção, o banco deve notificar o sistema e o departamento de contabilidade deverá enviar uma carta ao cliente.".

3.2.1. Identificação e especificação de requisitos

Como no processo da abordagem existem algumas actividades que podem ser realizadas em paralelo, tal como a identificação e especificação de *viewpoints* e *concerns*, estas foram encapsuladas numa caixa designada "Identificar e especificar" (ver figura 3.1).

3.2.1.1. Identificação de viewpoints e especificação de requisitos

A aplicação da abordagem integrada entre Problem Frames e AORE tem como ponto de partida, a identificação de cada *viewpoint* que desempenhe um papel preponderante no problema. Para cada *viewpoint* são também definidas e descritas as funcionalidades em que estes intervenham. Os *viewpoints* identificados no exemplo no exemplo de controlo de acesso a parques são os seguintes:

- *Cliente* – responsável por efectuar o registo e activar identificador.
- *Funcionário* – regista as reclamações.
- *Máquina do parque* – detecta o identificador do veículo, através de um sensor; em caso de sucesso abre a cancela.
- *Máquina de entrada* – sub-*viewpoint* da Máquina do parque, específico para controlar as entradas de veículos no parque.
- *Máquina de saída* – sub-*viewpoint* da Máquina do parque, específico para controlar as saídas de veículos do parque.
- *Identificador* – contém um número que deve ser detectado pelo sistema, por intermédio do sensor existente nas Máquinas entrada e saída do parque.
- *Banco* – realiza os débitos associados ao identificador e notifica o departamento de contabilidade em caso de insucesso no pagamento.
- *Departamento de Contabilidade* – envia uma carta ao cliente em caso de problemas na transacção.
- *ATM* – envia ao sistema a informação da transacção de associação do cartão de débito do cliente e identificador, permitindo a sua activação.
- *Administrador do sistema* – introduz e modifica informação no sistema.

Serão ilustrados os *viewpoints* Identificador, Máquina do parque e respectivos sub-*viewpoints* (Máquina de entrada e Máquina de saída) e Administrador do sistema, descritos utilizando o template da tabela 3.1. As tabelas 3.7 a 3.11 apresentam a descrição destes *viewpoints*.

Tabela 3.7: Template para *viewpoint* Identificador**.**

Atributos do Viewpoint	Descrição do Atributo
Nome:	Identificador (ID).
Foco:	Ser detectado pelo sensor da Máquina do parque.
Concerns:	Tempo de resposta, Disponibilidade, Correctude.
Fontes:	Sensor da Máquina do parque.
Requisitos:	R1 - O Identificador é detectado pelo sensor da Máquina do parque.

Tabela 3.8: Template para *viewpoint* Administrador do sistema**.**

Atributos do Viewpoint	Descrição do Atributo
Nome:	Administrador do sistema (AS).
Foco:	Responsável pela manutenção do sistema.
Concerns:	Disponibilidade, Correctude, Segurança.
Fontes:	Informação sobre parques de estacionamento.
Requisitos:	R1 - O Administrador do sistema gere a informação de um parque de estacionamento. R1.1 - O Administrador do sistema introduz um novo parque de estacionamento no sistema. R1.2 - O Administrador do sistema elimina um parque de estacionamento do sistema. R1.3 - O Administrador do sistema altera os preços de um parque de estacionamento no sistema. R2 - O Administrador do sistema trata a reclamação. R2.1 - O Administrador do sistema altera o estado da reclamação no sistema.

Tabela 3.9: Template para *viewpoint* Máquina do parque**.**

Atributos do Viewpoint	Descrição do Atributo
Nome:	Máquina do parque (MP).
Foco:	Responsável por detectar o Identificador através de um sensor, abrir e fechar a cancela.
Concerns:	Tempo de Resposta, Disponibilidade, Correctude, Multi-acesso.
Fontes:	Identificador, sensor.
Sub-VP:	Máquina de entrada, Máquina de saída.
Requisitos:	R1 - A Máquina do parque detecta o Identificador através do sensor e verifica no sistema se este se encontra no estado activo. R2 - A Máquina do parque abre a cancela. R2.1 - Se cancela da Máquina do parque estiver avariada, apresentar mensagem de erro “Cancela avariada”. R3 - A Máquina do parque fecha a cancela. R3.1 - Se cancela da Máquina do parque estiver avariada, apresentar mensagem de erro “Cancela avariada”.

Tabela 3.10: Template para *viewpoint* Máquina de entrada**.**

Atributos do Viewpoint	Descrição do Atributo
Nome:	Máquina de entrada (ME).
Foco:	Responsável por apresentar uma luz verde se o Identificador estiver activo, ou luz amarela no caso contrário.
Concerns:	Tempo de Resposta, Disponibilidade, Correctude, Multi-acesso.
Fontes:	Identificador, sensor.
Requisitos:	R1 - A Máquina de entrada apresenta uma luz verde se o Identificador está no estado activo. R2 - A Máquina de entrada apresenta uma luz amarela se o Identificador não está no estado activo. R3 - Se sensor da Máquina de entrada estiver avariado, apresentar mensagem de erro “Máquina avariada”. R4 - Se parque estiver com lotação máxima, apresentar mensagem de aviso “Parque esgotado”.

Tabela 3.11: Template para *viewpoint* Máquina de saída**.**

Atributos do Viewpoint	Descrição do Atributo
Nome:	Máquina de saída (MS).
Foco:	Responsável por apresentar a quantia a pagar.
Concerns:	Tempo de Resposta, Disponibilidade, Correctude, Multi-acesso.
Fontes:	Identificador, sensor.
Requisitos:	R1 - A Máquina de saída apresenta a quantia a pagar. R2 - Se sensor da Máquina de saída estiver avariado, apresentar mensagem de erro "Máquina avariada".

3.2.1.2. Identificação e especificação de concerns

No âmbito deste trabalho, cada *concern* diz respeito a um requisito não funcional do sistema. Estes podem ser obtidos, partindo da análise dos requisitos funcionais de cada *viewpoint* que por ele afectado, bem como, dos requisitos não funcionais que os clientes pretendam forçosamente ver implementados, como por exemplo a segurança ou o tempo de resposta de um sistema.

No exemplo do parque, em particular, foram detectados alguns *concerns* como Tempo de resposta, Disponibilidade, Segurança, Compatibilidade, Correctude e Multi-acesso. Para efeitos exemplificativos, optou-se apenas por apresentar a especificação do *concern* de Segurança, ver tabela 3.12. Para descrever cada *concern* emprega-se o template da tabela 3.2.

Tabela 3.12: Template para concern Segurança**.**

Atributos do *Concern*	Descrição do Atributo
Nome:	Segurança
Descrição:	Garantir a segurança dos dados e controlar o acesso ao sistema.
Viewpoints influenciados:	Cliente, Funcionário, ATM, Administrador do sistema, Banco.
Requisitos:	R1 - O sistema deve providenciar segurança para: R1.1 - Os dados de registo fornecidos pelo Cliente através da ATM. R1.2 - Controlar o acesso ao sistema ao Administrador do sistema através de palavra-chave. R1.3 - Controlar o acesso ao sistema ao Funcionário do parque. R1.4 - Os dados das transacções de pagamento com o Banco.

A especificação de todos os *viewpoints* e *concerns* do problema possibilita iniciar a detecção dos elementos aspectuais funcionais e não funcionais. No entanto, também permite

obter todos os dados necessários para elaborar o Diagrama de Contexto, no âmbito de Problem Frames, dado que cada *viewpoint* identificado pode vir a corresponder a um novo domínio no problema. Também a análise dos requisitos de cada um dos *viewpoints* facilita a percepção dos fenómenos presentes e que são partilhados entre os domínios.

3.2.2. Identificação de aspectos não funcionais

A modularização de cada um dos aspectos não funcionais de um problema diz respeito, unicamente, à detecção das relações *crosscutting* que os *concerns* possuam com qualquer *viewpoint* existente.

3.2.2.1. Relacionar viewpoints e concerns

Após ter sido efectuada a identificação e especificação de todos os *viewpoints* e *concerns*, torna-se possível realizar o seu relacionamento. A tabela 3.13 é respeitante à relação entre *concerns* e *viewpoints*.

Tabela 3.13: Matriz de relacionamento entre *concerns* e *viewpoints*.

Viewpoints / Concerns	MP	ME	MS	Cliente	Funcionário	Banco	ATM	DC	ID	AS
Tempo de resposta	X	X	X						X	
Disponibilidade	X	X	X		X		X		X	X
Compatibilidade						X	X			
Correctude	X	X	X	X	X	X	X	X	X	X
Multi-acesso	X	X	X				X		X	
Segurança				X	X	X	X			X

(MP: Máquina do parque**; ME:** Máquina de entrada**; MS:** Máquina de saída**; DC:** Departamento de contabilidade**; ID:** Identificador**; AS:** Administrador do sistema**)**

Através deste relacionamento são detectados os *crosscutting concerns* não funcionais, isto é, todos aqueles que afectem mais do que um *viewpoint* no seu funcionamento.

3.2.2.2. Identificar crosscutting concerns

Neste exemplo, e como é ilustrado na tabela 3.13, todos os *concerns* acabam por afectar mais de que um *viewpoint*. Estes serão considerados os aspectos não funcionais do

problema, como é o caso da Disponibilidade, que afecta os *viewpoints* Máquina do parque, ATM, Administrador do sistema, Funcionário e Identificador.

3.2.2.3. Identificação e resolução de conflitos

No caso particular dos requisitos não funcionais, estes podem, por vezes, limitar a correcta implementação de outros, tornando-se essencial decidir quais serão tidos em maior consideração no âmbito do problema. Através da elaboração de uma matriz de contribuições entre aspectos, define-se o modo como cada aspecto pode afectar positivamente ou negativamente outro. A tabela 3.14 demonstra, por exemplo, como o aspecto Tempo de resposta influencia negativamente Correctude ou Segurança, originando assim, uma possível situação de conflito sempre que estes aspectos sejam necessários num mesmo *viewpoint*.

Tabela 3.14: Matriz de contribuições entre aspectos**.**

Aspectos / Aspectos	Tempo de resposta	Disponibilidade	Compatibilidade	Correctude	Multi-acesso	Segurança
Tempo de resposta		+		-	+	-
Disponibilidade	+				+	
Compatibilidade	+	+		+	+	+
Correctude	-	+				+
Multi-acesso	-	-				
Segurança	-	+	-	+		

A partir da derivação da matriz da tabela 3.13, é agora imprescindível atribuir prioridades (valores no intervalo de [0..1]) a cada aspecto, dependendo do grau de importância definido para cada um, isto no caso de estarem presentes aspectos conflituosos. A matriz da tabela 3.15 ilustra a identificação desses mesmos conflitos, e sua consequente resolução por intermédio da aplicação das prioridades a cada aspecto.

Tabela 3.15: Matriz de prioridades em *viewpoints* entre aspectos conflituosos**.**

Viewpoints / Aspectos	MP	ME	MS	Cliente	Funcionário	Banco	ATM	DC	ID	AS
Tempo de resposta	0.9	0.9	0.9						0.9	
Disponibilidade	0.6	0.6	0.6		X		0.8		0.8	X
Compatibilidade						0.7	0.6			
Correctude	0.7	0.7	0.7	X	X	X	X	X	0.7	X
Multi-acesso	1.0	1.0	1.0				1.0		1.0	
Segurança				X	X	1.0	0.9			X

(MP: Máquina do parque**; ME:** Máquina de entrada**; MS:** Máquina de saída**; DC:** Departamento de contabilidade**; ID:** Identificador**; AS:** Administrador do sistema**)**

Após análise da matriz da tabela 3.15, é possível verificar como foi solucionado, por exemplo, o conflito existente entre Multi-accesso e Disponibilidade ou Correctude e Tempo de resposta no *viewpoint* Máquina do parque.

3.2.3. Identificação de aspectos funcionais

De modo a determinar os aspectos funcionais do problema, é necessário primeiramente identificar as diversas e principais funcionalidades que o sistema deve ter implementado. Para tal, torna-se indispensável efectuar uma análise cuidada de todos os requisitos de cada *viewpoint*.

3.2.3.1. Identificação de funcionalidades e requisitos

Após a análise aos requisitos de cada *viewpoint*, pode-se concluir que neste exemplo existiriam funcionalidades tais como entrar/sair do parque, activar o identificador, registar reclamação ou efectuar débito semanal.

3.2.3.2. Relacionar viewpoints, requisitos e funcionalidades

Cada funcionalidade pode possuir um ou mais requisitos que necessariamente terão de ser alcançados, podendo envolver mais do que apenas um *viewpoint*. Para se verificar quais dos requisitos são *crosscutting*, e à semelhança do método que foi utilizado para os aspectos não funcionais, relaciona-se, por intermédio de uma matriz, cada requisito de um determinado *viewpoint* com as funcionalidades que lhe são respeitantes.

Para exemplificar a detecção de aspectos funcionais, vão ser tomadas como exemplos, as principais funcionalidades do sistema, Entrar no parque e Sair do parque. A primeira é constituída, após a análise aos requisitos dos *viewpoints* intervenientes, por requisitos tais como, detectar o Identificador, abrir a cancela, mostrar luz verde e fechar a cancela. Também a funcionalidade de Sair do parque, como contém alguns destes requisitos permite demonstrar com relativa facilidade este método para detectar aspectos funcionais. Na tabela 3.16 encontra-se representada a matriz de relacionamentos entre os requisitos de cada *viewpoint* descrito na secção 3.2.1.1 e as funcionalidades a eles pertencentes.

Tabela 3.16: Matriz de relacionamento entre *viewpoints*, requisitos e funcionalidades**.**

Funcionalidades / Viewpoints	Entrar no parque	Sair do parque	Gerir parque
	Requisitos de *Viewpoints*		
ID	ID.R1	ID.R1	
MP	MP.R1; MP.R2; MP.R3;	MP.R1; MP.R2; MP.R3;	
ME	ME.R1; ME.R2; ME.R3; ME.R4		
MS		MS.R1; MS.R2	
AS			AS.R1

(MP: Máquina do parque**; ME:** Máquina de entrada**; MS:** Máquina de saída**; ID:** Identificador; **AS:** Administrador do sistema**)**

3.2.3.3. Identificar aspectos funcionais

Como ilustrado na tabela 3.16, os requisitos dos *viewpoints* Identificador (ID.R1) e Máquina do parque (MP.R1, MP.R2 e MP.R3) podem ser automaticamente identificados e designados *crosscutting*, pois existe a necessidade de os alcançar, tanto na funcionalidade de Entrar no parque, bem como em Sair do parque. O mesmo não sucede com os requisitos dos *viewpoints* Máquina de entrada e Máquina de saída do parque, porque apenas ocorrem em uma das duas funcionalidades, daí que não se considerem como *crosscutting*. No entanto, após uma observação cuidada, verifica-se que o requisito ME.R3 da Máquina de entrada e MS.R2 da Máquina de saída são de facto semelhantes, embora se encontrem em funcionalidades distintas, diferem apenas no nome dos *viewpoints* em causa (sub-*viewpoints* neste caso em particular). Em [23], a existência deste tipo de requisitos em diferentes *viewpoints*, permite definir *viewpoints* aspectuais, que não é de todo o objectivo deste trabalho. Mas possibilita um meio de separar os requisitos funcionais similares aspectuais, para posterior representação em problemas. Portanto, sempre que dois ou mais requisitos semelhantes sejam provenientes de *viewpoints* distintos, se verifiquem numa ou mais

funcionalidades, são considerados aspectos funcionais. Desta forma, os requisitos de *viewpoints* (ID.R1, MP.R1, MP.R2, MP.R3 e ME.R3-MS.R2) *crosscutting* vão originar diversos aspectos funcionais que deverão ser modularizados e especificados directamente através da representação de Problem Frames, originando respectivamente, um diagrama de problema aspectual.

É também indispensável definir já quais os aspectos que foram identificados, para que possa ser realizada a correspondência com o diagrama de problema aspectual respectivo. Neste caso em concreto, cada requisito aspectual vai corresponder a um aspecto funcional, à excepção do aspecto Detectar identificador (AspDetectarId), composto pelos requisitos ID.R1 e MP.R1, e Apresentar mensagem de erro (AspApresentarMsgErro), composto por ME.R3 e MS.R2, pois dizem respeito a um requisito semelhante aspectual, podendo assim serem agrupados. A tabela 3.17 ilustra os aspectos funcionais detectados e os requisitos que os constituem.

Tabela 3.17: Aspectos funcionais**.**

Aspecto funcional	Requisitos
AspDetectarId	(ID.R1 - MP.R1) – Detectar Identificador
AspAbrirCancela	MP.R2 – Abrir cancela do parque
AspFecharCancela	MP.R3 – Fechar cancela do parque
AspApresentarMsgErro	(ME.R3 - MS.R2) – Apresentar mensagem de erro

Com a especificação dos aspectos funcionais, como Abrir a cancela (AspAbrirCancela) do exemplo da tabela 3.17, os requisitos que compõem cada *viewpoint*, devem ser removidos de ambas as funcionalidades da tabela 3.16, originando uma nova tabela. Esta encontra-se ilustrada abaixo na tabela 3.18 e apresenta as funcionalidades compostas exclusivamente pelos seus requisitos não aspectuais.

Tabela 3.18: Matriz de relacionamento entre funcionalidades e requisitos de *viewpoints* com remoção de aspectos funcionais**.**

Funcionalidades / Viewpoints	Entrar no parque	Sair do parque	Gerir parque
	Requisitos de *Viewpoints*		
ME	ME.R1; ME.R2; ME.R4		
MS		MS.R1	
AS			AS.R1

(ME: Máquina de entrada**; MS:** Máquina de saída**; AS:** Administrador do sistema**)**

Após a remoção dos aspectos das funcionalidades, estes já se encontram disponíveis para serem representados e modularizados através de um diagrama de problema aspectual,

como será demonstrado na secção 3.2.4.3. No caso de cada funcionalidade, e como já não possui qualquer aspecto associado, a sua representação será elaborada através de diagramas de problema (secção 3.2.4.2).

3.2.4. Problem Frames

Esta secção tem como objectivo explicitar, por intermédio do exemplo de controlo de acesso a parques, como é feita a utilização e integração de Problem Frames na abordagem proposta. Cada funcionalidade, aspecto ou concern será representado através de um diagrama de problema. No entanto, primeiro, é necessário contextualizar o problema em causa através de um diagrama de contexto.

3.2.4.1. Diagrama de Contexto

O diagrama de contexto marca o início da utilização da abordagem de Problem Frames para especificação dos requisitos funcionais e não funcionais, anteriormente identificados, do sistema que se pretende desenvolver.

O Diagrama de Contexto procura compreender o contexto em que o problema de software se insere. É desta forma, que a abordagem apresentada neste trabalho completa esse processo, procurando relacioná-lo com o modelo AORE baseado em *viewpoints*, em que cada domínio do diagrama de contexto é obtido a partir da análise das descrições já efectuadas sobre cada *viewpoint*. Desse modo, torna-se possível identificar os diversos domínios do mundo real que interagem verdadeiramente com o sistema.

Cada *viewpoint* por si só pode, na maioria dos casos, vir ter correspondência directa com um domínio. Nos casos em que existam sub-*viewpoints*, apenas estes irão figurar no diagrama de contexto, já que herdam do super *viewpoint* os seus requisitos.

Neste caso, para além de todos os *viewpoints* mencionados anteriormente na secção 3.2.1, existem outros domínios do problema em causa, que são também necessários considerar.

O diagrama de contexto do problema deste caso de estudo em particular é ilustrado na figura 3.2. Analisando o exemplo do parque, mais concretamente o requisito do *viewpoint* Máquina do parque e respectivos sub-*viewpoints* (Máquina de entrada e Máquina de saída), é possível extrair ainda, como elementos relevantes do problema, os domínios Cancela, Painel

de luzes, Indicador da quantia e Mostrador de mensagem. O mesmo sucede com os domínios de bases de dados Clientes, Identificadores, Parques, Utilizadores e Reclamações, provenientes da observação dos *viewpoints* Cliente, Admistrador do sistema e Funcionário. É de realçar o *viewpoint* Máquina do parque, composto por dois sub-*viewpoints*, onde cada um dará assim origem, a dois novos domínios (Máquina de entrada e Máquina de saída).

Para efeitos de simplificação, os fenómenos partilhados entre os domínios serão representados nos diagramas de problema respectivos.

Como apresentado na figura 3.2, os domínios Identificadores, Clientes, Reclamações, Utilizadores e Parques correspondem a partes do mundo do problema que têm de ser estruturadas e implementadas. Neste caso representam sistemas de bases de dados. O domínio máquina, identificado pelas duas riscas, é representado por Máquina do Parque de Estacionamento (MPE). Todas as outras interacções entre os domínios do problema e a máquina derivam dos requisitos funcionais de cada *viewpoint* e que se desejam ver implementados.

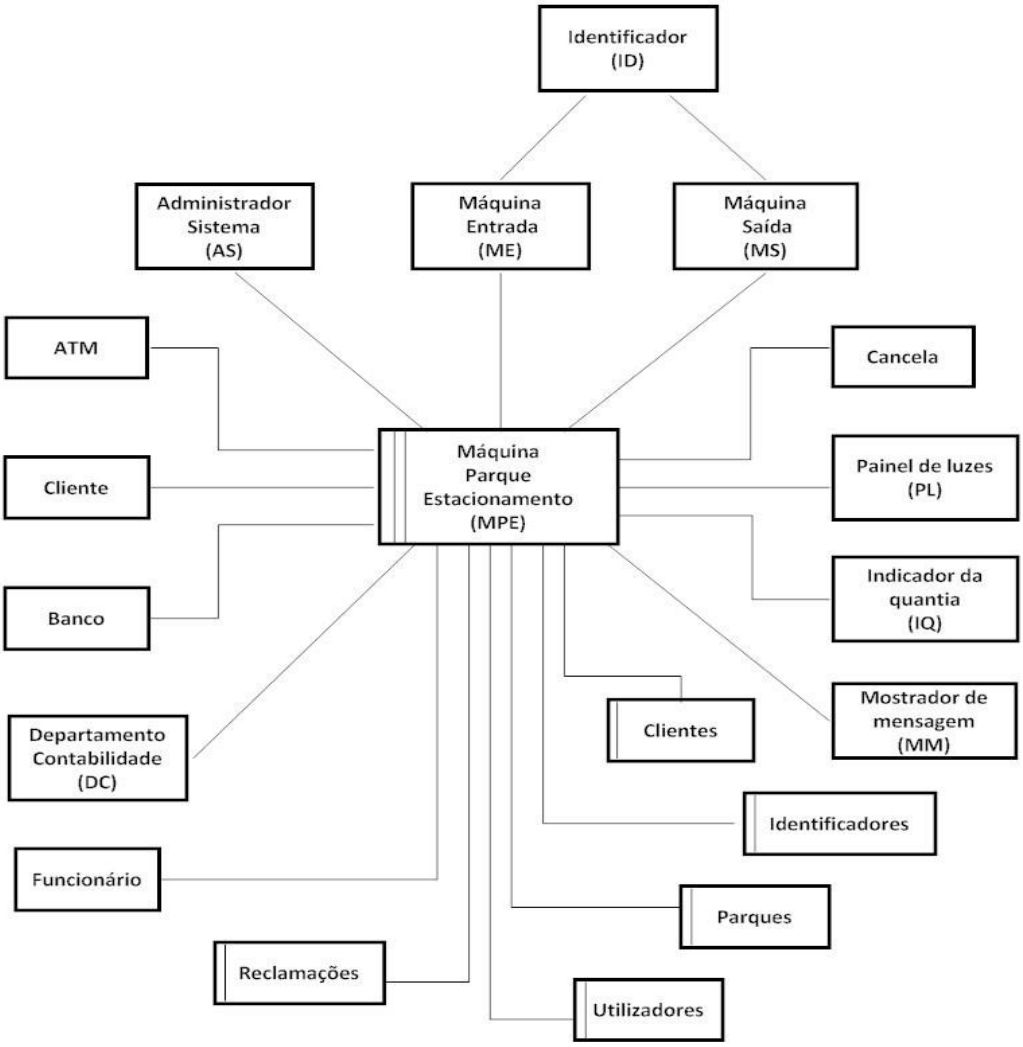


Figura 3.2: Diagrama de Contexto**.**

3.2.4.2. Diagramas de problema

Cada diagrama de problema vai derivar de um ou mais requisitos provenientes dos *viewpoints*. O mesmo sucede com os fenómenos partilhados entre os domínios, pois também eles serão determinados através da sua análise. No entanto, cada diagrama de problema ilustrado, procura representar cada funcionalidade, embora não inclua os diversos requisitos pertencentes aos aspectos funcionais detectados anteriormente, restando-lhe apenas os requisitos não aspectuais. É dessa forma, que o domínio máquina de cada diagrama mantém a designação com o nome identificativo da funcionalidade principal e não dos sub-requisitos que a compõem. Na secção 3.2.5, aquando é realizada a decomposição de cada problema em problemas de menores dimensões, o domínio máquina já adopta a denominação de acordo com o requisito em causa.

Na figura 3.3 é descrito com maior detalhe o diagrama de problema respectivo à funcionalidade de Entrar no parque sem aspectos.

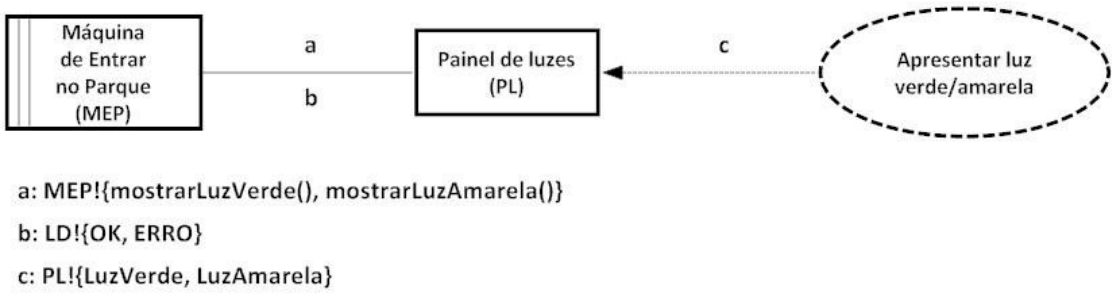


Figura 3.3: Diagrama de problema Entrar no parque**.**

Esta figura demonstra as interacções existentes entre os domínios para que os requisitos de mostrar a luz verde ou amarela aquando a entrada no parque. Os fenómenos de interacção encontram-se representados pelas letras. Nesta situação, a Máquina de Entrar no Parque (MEP), que diz respeito à funcionalidade de entrar no parque, daí o seu nome, fica responsável pelos requisitos remanescentes (apresentar luz verde ou amarela), após a remoção dos requisitos aspectuais. Esta efectua o pedido para que seja apresentada a luz verde sensor, caso o sensor detecte o Identificador com sucesso e este se encontre activo, ou luz amarela no caso contrário. Assim serão atingidos os estados objectivos (luz verde ou amarela dependendo da situação) e os respectivos requisitos que se pretendiam alcançar no início.

No que toca à funcionalidade de gerir informações de um parque através do Administrador do sistema, a figura 3.4 caracteriza o diagrama de um sub-problema da funcionalidade Gerir parque, mais concretamente, o requisito de alterar o preço.

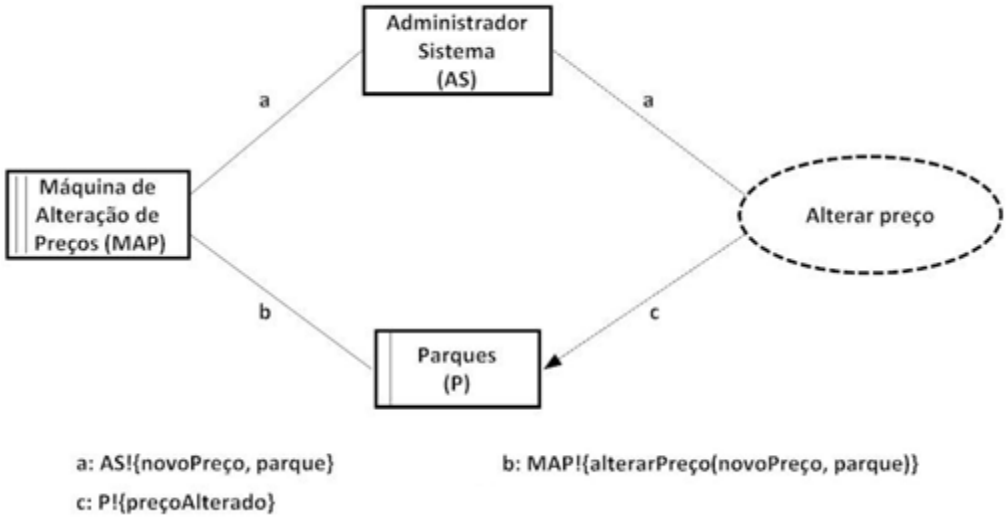


Figura 3.4: Diagrama de problema Alterar preço**.**

Como se pode observar através da figura 3.4, para alcançar com sucesso o requisito de alterar informações de um parque, como o preço, é necessário que a Máquina de alterar preço receba por parte do administrador do sistema os dados indispensáveis, como um novo preço e o parque em que este entrará em vigor. Desse modo, e para que o preço no parque em causa passe à condição de alterado, é fundamental modificar o seu valor na base de dados de Parques. Outros requisitos poderiam aqui também ser representados, tais como, a criação ou remoção de um novo parque.

3.2.4.3. Diagramas de problema aspectuais

Neste passo do processo, os aspectos funcionais, obtidos na análise dos requisitos de cada *viewpoint* como mencionado na secção 3.2.3, vão ser representados por intermédio de Diagramas de Problema aspectuais, em que apenas os elementos que possam ser variáveis serão representados por domínios ou fenómenos abstractos. Isto permitirá fazer a correspondência com qualquer problema em que o aspecto funcional seja utilizado. Para exemplificar esta técnica, vai ser tomado o aspecto funcional Abrir a cancela (AspAbrirCancela), anteriormente demonstrado na tabela 3.17, indispensável para que os objectivos de entrar e sair do parque sejam obtidos. Os domínios e fenómenos que sejam exclusivamente utilizados para este aspecto devem ser removidos dos diagramas de problema

correspondentes às funcionalidades de Entrar e Sair do parque. O diagrama de problema aspectual da figura 3.5 representa, portanto, o aspecto Abrir a cancela. Cada diagrama aspectual deve ser primeiro representado individualmente.

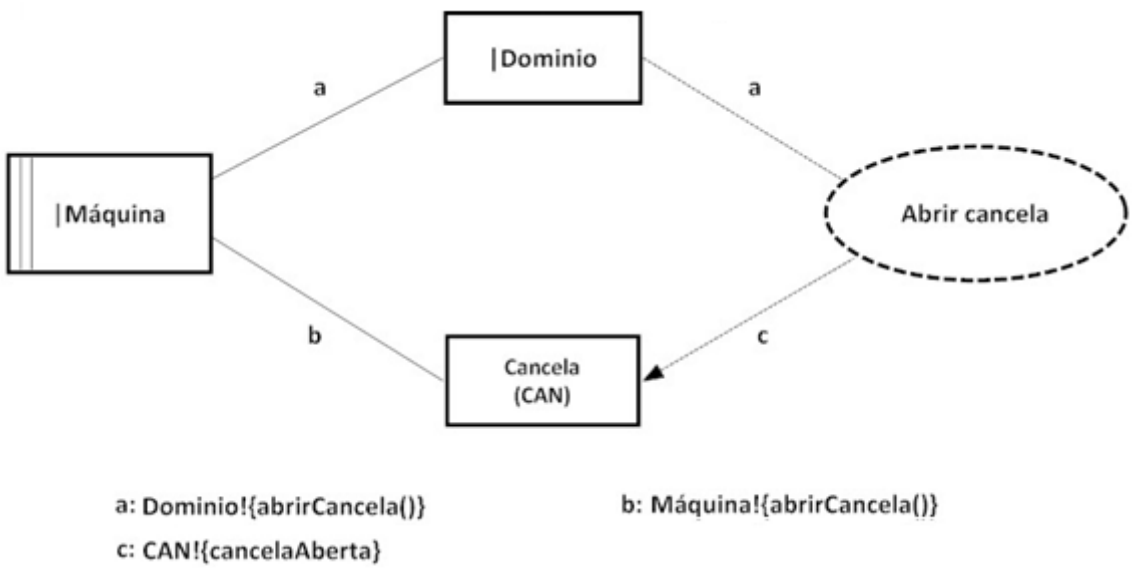


Figura 3.5: Diagrama de problema aspectual Abrir a cancela (AspAbrirCancela)**.**

O diagrama da figura 3.5 representa, portanto, os sub-requisitos funcionais aspectuais da funcionalidade de Entrar e Sair do parque. Este padrão é utilizado sempre que uma determinada funcionalidade do sistema exija um dos requisitos mencionados. A variável |Dominio desempenha o papel de um qualquer domínio que verifique as mesmas condições de interacção ou de partilha de fenómenos nas diferentes ocorrências. Neste caso, |Dominio poderia representar os domínios Máquina de entrada (ME) e Máquina de saída (MS), consoante o problema (Entrar ou Sair do parque) que se pretende figurar. O mesmo sucede com o domínio máquina |Máquina, que virá a ser instanciado de acordo com a funcionalidade em causa. No entanto, a especificação de um aspecto, quer seja este não funcional ou funcional, não pode ficar por aqui, pois resta ainda definir uma regra de composição que possa descrever quais as partes do problema inicial por ele afectados e que restabeleça as ligações com cada componente envolvida.

3.2.4.4. Operacionalização de aspectos não funcionais

Alguns dos requisitos não funcionais que se desejam ver presentes no funcionamento do sistema, podem ser, para além da sua descrição em função a *viewpoints*, especificados por intermédio de diagramas de problema. Assim, e como a grande parte, senão mesmo a

totalidade dos *concerns* identificados num problema de software são considerados aspectos, estes podem, deste modo, ser incorporados na sua representação em Problem Frames.

A operacionalização [7] de cada aspecto tem como fim demonstrar que existe uma máquina que pode ser implementada de forma a garantir que este requisito se cumpra. A operacionalização de cada aspecto não funcional, não é limitada apenas com operações ou funções, mas também com a introdução de restrições. Permite assim representar de um modo concreto, os domínios envolvidos e as interacções necessárias para que o requisito não funcional pretendido seja alcançado, explicando a causa da sua necessidade em relação a, por exemplo, um *viewpoint*.

No entanto, nem todos os *concerns* se podem considerar operacionalizáveis, isto é, alguns podem ser encarados como demasiado subjectivos (como por exemplo o baixo custo, a satisfação ou qualidade) para serem ilustrados e descritos recorrendo aos diagramas de Problem Frames.

De modo a exemplificar este método de operacionalização de um requisito não funcional aspectual, vai se tomar o requisito 1.2 do aspecto Segurança, identificado na tabela 3.12.

O diagrama de problema Controlar acesso da figura 3.6 ilustra então a necessidade de garantir um acesso seguro às bases de dados que vão permitir a manutenção da informação respeitante aos parques, ou seja, é um requisito não funcional aspectual de segurança aplicável, por exemplo, ao problema Alterar preço da figura 3.4.

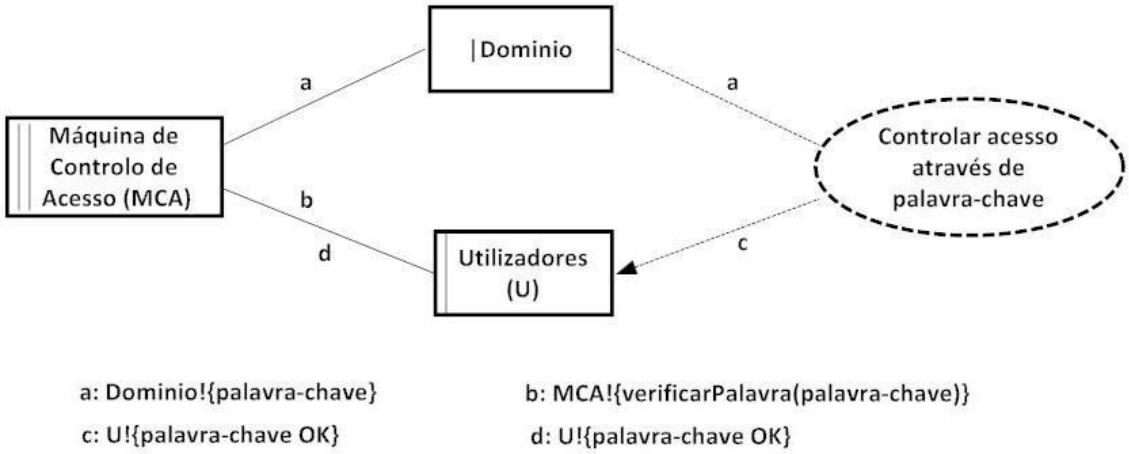


Figura 3.6: Diagrama de problema aspectual não funcional Controlar acesso**.**

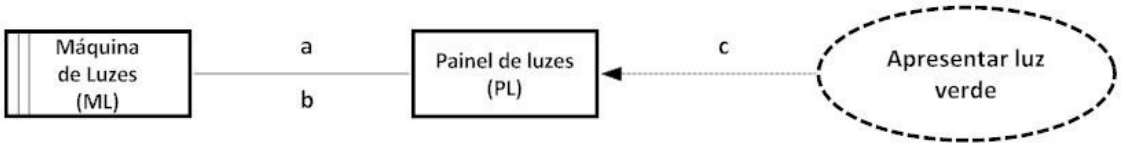
Através da figura 3.6 é possível verificar que o requisito de segurança respeitante ao controlo de acesso ao sistema, pode ser obtido por intermédio da Máquina de controlo de

acesso (MCA), recebendo como input uma palavra-chave e garantindo assim o acesso a um utilizador. O domínio |Dominio será posteriormente instanciado a Administrador do sistema através das regras de composição de aspectos não funcionais.

Deste modo é demonstrado o processo de operacionalização e especificação de um requisito não funcional aspectual, utilizando para isso, o modelo de Problem Frames. A composição deste tipo de diagrama de problema com os problemas originais é descrita na secção 3.2.5.2.

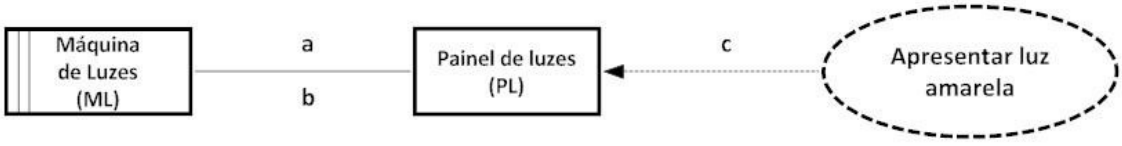
3.2.4.5. Decomposição e especificação de problemas

A decomposição e consequente divisão de problemas maiores em problemas menores, vai permitir uma análise mais correcta de cada sub-problema com uma maior coerência na sua descrição. Se tomarmos como exemplo o diagrama de problema Entrar no parque da figura 3.3, este pode ainda ser decomposto em dois novos sub-problemas, realizando-se uma separação dos seus requisitos apresentar a luz verde e amarela. As figuras 3.7 e 3.8 representam os novos diagramas dos sub-problemas de Entrar no parque, Apresentar a luz verde e luz amarela.



a: ML!{mostrarLuzVerde()}
b: LD!{OK, ERRO}
c: PL!{LuzVerde}

Figura 3.7: Diagrama de problema Apresentar luz verde**.**



a: ML!{mostrarLuzAmarela()}
b: LD!{OK, ERRO}
c: PL!{LuzAmarela}

Figura 3.8: Diagrama de problema Apresentar luz amarela**.**

Se considerarmos que cada diagrama já se encontra na sua forma mais simples, é então dessa forma que é iniciada a utilização dos *problem frames* elementares [13] no auxílio da sua descrição.

Problem frames elementares são classes de problemas já conhecidas e documentadas que servem como padrões para a análise, categorização e caracterização de problemas de software.

No caso particular do problema da figura 3.7, este pode ser associado ao Required Behaviour. O mesmo sucede com a figura 3.8. Já na figura 3.4, o diagrama de Alterar preço pertence a uma outra classe padrão de problemas (*Simple Workpieces*), onde é necessário alterar algum tipo de domínio léxico, nomeadamente, a base de dados dos parques de estacionamento.

Dependendo do tipo de *problem frame* que se encontra relacionado com o problema em causa, vão também depender as definições a que este estará sujeito, na sua especificação através do *frame concern* [13].

As figuras 3.9 e 3.10 demonstram a utilização do *frame concern* recorrendo ao uso de descrições que complementem a explicação do diagrama Alterar preço. Esta argumentação do *frame concern* encontra-se ordenada pela sequência de eventos que permitem atingir o objectivo pretendido.

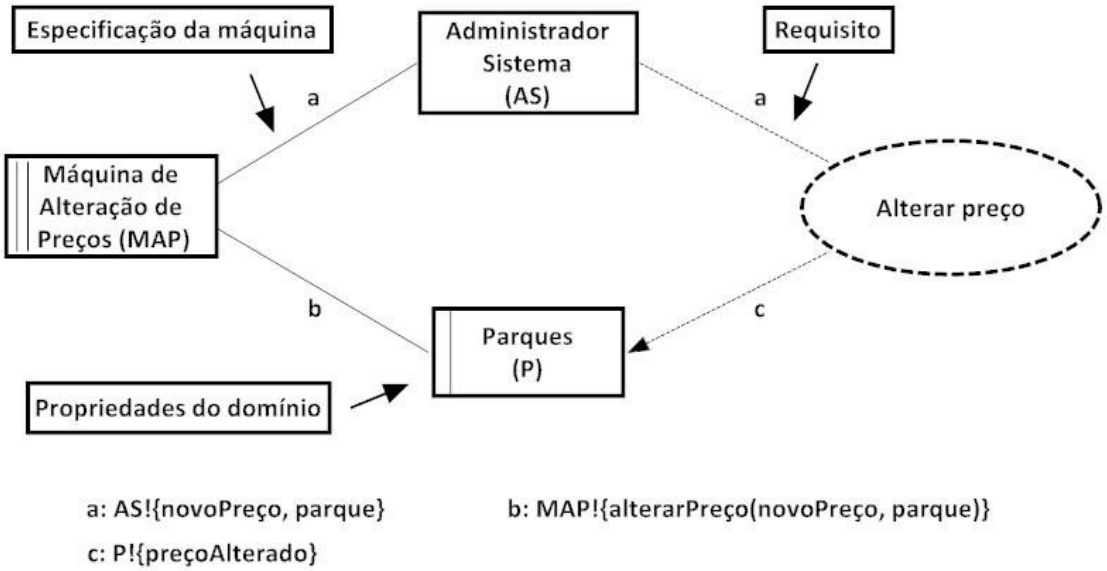


Figura 3.9: Diagrama Simple Workpieces *problem frame* com *frame concern*.

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o Administrador do sistema introduz um novo preço e o parque na MAP.	Requisito
2. A MAP acede à base de dados Parques e efectua o comando alterarPreço(novoPreço, parque).	Especificação da máquina
3. Resultando na alteração do valor do preço no parque.	Propriedades do dominio
4. Desta forma é atingido o requisito de Alterar preço.	Requisito

Figura 3.10: Descrições do *frame concern*.

Na próxima secção vai ser demonstrada a forma de reagrupar as funcionalidades e os requisitos aspectuais (funcionais e não funcionais), por intermédio da utilização de regras de composição de aspectos.

3.2.5. Regras de composição de aspectos

A modularização de cada elemento aspectual do problema que se encontra sob análise requer que estejam encontrados todos os requisitos, funcionais e não funcionais, de natureza *crosscutting*. Como neste ponto, já todos eles estão perfeitamente identificados e especificados, tendo para isso contribuído a abordagem AORE [22] sustentada em *viewpoints* e Problem Frames [13], é o momento de aplicar as regras de composição para cada aspecto.

Estas regras têm como principal função integrar e compor novamente as ligações perdidas entre cada componente aspectual reconhecida e as funcionalidades correspondentes. Esta composição tira proveito do facto de cada elemento (domínio, fenómeno ou requisito) se encontrar já representado graficamente através dos diagramas de problema, o que simplifica a sua constituição.

3.2.5.1. Regras de composição de aspectos funcionais

No caso dos aspectos funcionais, estes serão novamente integrados com os diagramas de problemas respectivos às funcionalidades ao qual foram retirados. A regra de composição de diagramas de problema aspectuais, baseia-se essencialmente em *pattern matching* [28] e numa linguagem que tem como objectivo restabelecer as ligações entre domínios e seus fenómenos partilhados. A figura 3.11 demonstra a sintaxe da regra de composição de um

requisito aspectual funcional. A figura 3.12 aplica essa mesma regra, instanciando-a com o aspecto funcional Abrir a cancela (AspAbrirCancela) ao problema Entrar no parque.

```
Compose aspect <aspectualProblemDiagram name> with <problem name>
Bind domain <aspectdomain name> to <domain name>
{ Bind domain <aspectdomain name> to <domain name> }
{ Bind phenomenon <aspectphenomenon name> to <phenomenon name> }
```

Figura 3.11: Regra de composição de um diagrama de problema aspectual**.**

```
Compose aspect AspAbrirCancela with Entrar no parque
Bind domain |Máquina to Máquina de Entrar no parque
Bind domain |Dominio to Máquina de Entrada
```

Figura 3.12: Instanciação do diagrama de problema aspectual Abrir a cancela**.**

A aplicação da regra de composição do diagrama de problema aspectual é aplicada sempre que se pretende integrar esse mesmo aspecto com o problema inicial e vai descrever o problema que este afecta, os domínios aspectuais envolvidos e os requisitos existentes. Estas descrições são obtidas através da utilização de palavras como **compose** (composição entre digramas de problema aspectuais e não aspectuais) e **bind**, que representa a instanciação de domínios, aqui designados por |Máquina ou |Dominio, aos domínios reais do problema, no caso do exemplo, a Máquina de Entrar no Parque (MEP) e Máquina de entrada (ME) respectivamente. **Bind** pode também servir para instanciar fenómenos presentes no diagrama de problema aspectual, que possam tomar conteúdos distintos nas diferentes funcionalidades.

No entanto, o problema inicial Entrar no parque tem também de ser composto com os restantes aspectos identificados inicialmente, ou seja, Detectar identificador (AspDetectarId), Fechar a cancela (AspFecharCancela) e Apresentar mensagem de erro (AspApresentarMsgErro).

Após a composição de todos os diagramas, o problema Entrar no parque ficaria então novamente completo, como é ilustrado na figura 3.13.

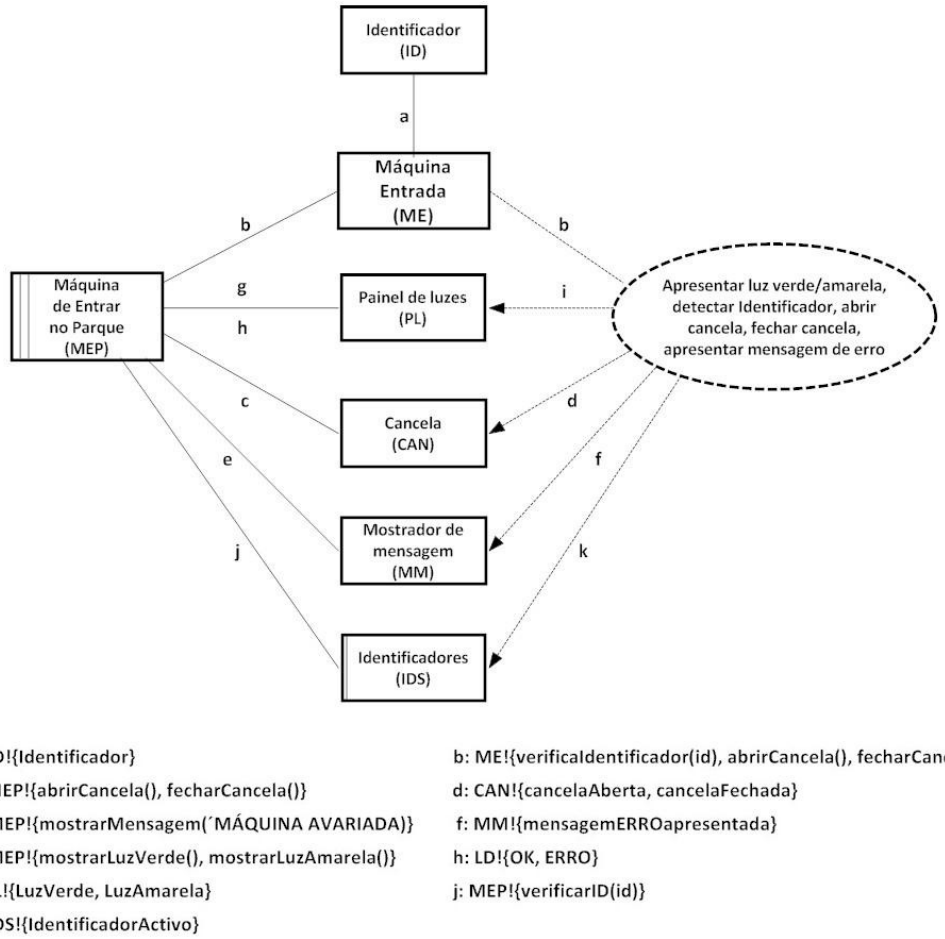


Figura 3.13: Diagrama de problema Entrar no parque composto**.**

De seguida, apresentar-se-á a composição de um aspecto não funcional a um problema.

3.2.5.2. Regras de composição de aspectos não funcionais

A composição de aspectos não funcionais é efectuada integrando no diagrama de problema afectado pelo aspecto não funcional, o diagrama de problema operacionalizado que lhe corresponde.

A regra de composição de um aspecto não funcional é a mesma utilizada nos aspectos funcionais e está presente na figura 3.11. O exemplo da sua instanciação, no caso do problema do administrador do sistema alterar preços de um parque é ilustrada na figura 3.14.

```
Compose aspect Controlar acesso with Alterar preço
Bind domain |Dominio to Administrador do sistema
```

Figura 3.14: Instanciação do diagrama de problema Alterar preço com controlo de acesso**.**

O exemplo do problema Alterar preço com controlo de acesso, ilustrado na figura 3.15, demonstra então a composição final entre os problemas Alterar preço (figura 3.4) e Controlar acesso (figura 3.6). Este diagrama representa o requisito não funcional, do *viewpoint* Administrador do sistema, em que este é responsável pela sua manutenção, já tendo em consideração a segurança de acesso através de uma palavra-chave.

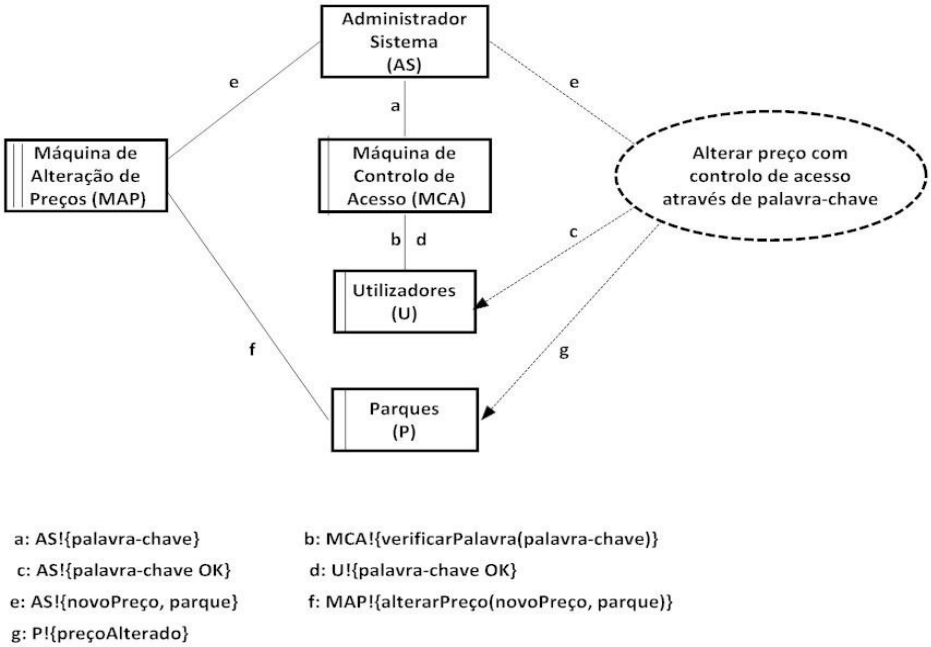


Figura 3.15: Diagrama de problema composto Alterar preço com controlo de acesso**.**

Na figura 3.15 são apresentadas as interacções compostas entre os domínios já anteriormente observadas nos problemas individuais Alterar preço e Controlar acesso. Desta forma se evidencia a necessidade de recorrer ao modelo de especificação de Problem Frames, para o caso particular dos requisitos não funcionais. Este método permite uma descrição mais detalhada e completa de cada problema, possibilitando incluir os *concerns* a ele associados.

3.3. Meta-modelo da abordagem

O meta-modelo de Problem Frames proposto nesta dissertação, toma como ponto de partida os meta-modelos de Problem Frames anteriormente apresentados em [18] e [11], sendo complementado com a integração de conceitos orientados a aspectos provenientes de uma abordagem AORE baseada em *viewpoints* [22].

Os modelos são representados por diagramas de classes UML [27]. Uma classe pode relacionar-se com outras de diversas formas – generalização, associação, agregação ou composição - podendo exibir atributos ou métodos. Em UML, cada associação é representada como uma linha que interliga as classes participantes do relacionamento, podendo também mostrar a regra e a multiplicidade de cada um. Esta multiplicidade é exibida como um intervalo [min..máx] de valores não negativos, com uma estrela (*) no lado máximo representando infinito. As extremidades da cada associação contém nomes para futura especificação de expressões OCL, tal como em [11].

De modo a refinar e descrever restrições adicionais a um modelo UML, o OCL [11, 20] consegue providenciar todos os meios relevantes para uma descrição muito mais correcta, evitando também qualquer ambiguidade. Esta linguagem formal de pura especificação não altera em nada o que se encontra representado pelo modelo, daí que um estado do sistema nunca seja alterado devido à avaliação de uma qualquer expressão. O OCL é uma linguagem essencialmente de modelação e não de programação.

3.3.1. Meta-modelo AORE baseado em viewpoints

A figura 3.16 apresenta um diagrama de classes UML, de maneira a representar o meta-modelo de uma técnica AORE sustentada em *viewpoints* [22].

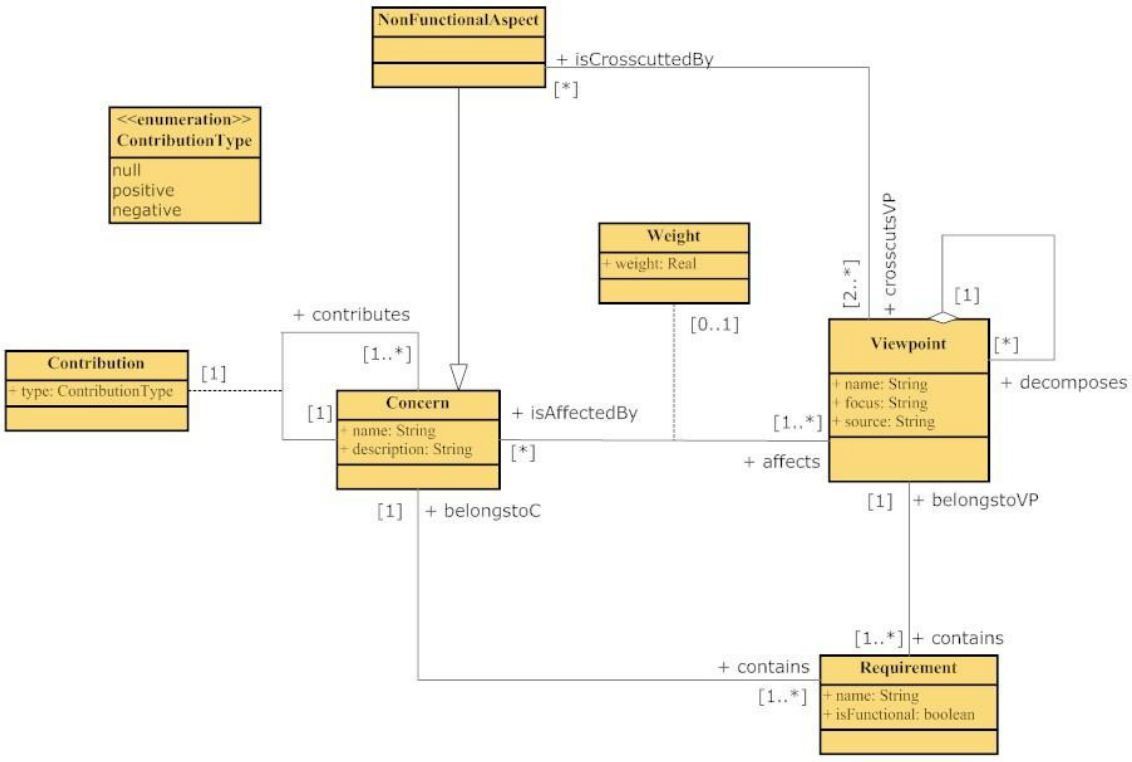


Figura 3.16: Meta-modelo AORE com *viewpoints*.

O modelo ilustrado na figura 3.16 permite verificar como se relacionam os vários elementos de uma técnica AORE baseada em *viewpoints*, como apresentada em [22].

Um Viewpoint é a componente que possui uma determinada perspectiva sobre o sistema, e como tal, um papel de extrema importância para o seu correcto funcionamento. Cada Viewpoint é caracterizado por atributos como o seu nome, um foco e fonte. Este elemento vai conter diversos requisitos funcionais no qual participa activamente, daí que contenha uma associação com a classe Requirement (contains com multiplicidade [1..*]).

Como foi possível constatar ao longo da descrição da abordagem na secção anterior, um Viewpoint pode ser decomposto em mais do que um sub-*viewpoint*, ou seja, possui uma relação *decomposes* com multiplicidade [*] com ele próprio. Em termos de requisitos não funcionais (*concerns*), cada Viewpoint pode ser afectado por nenhum, um ou diversos Concern, pelo que a sua relação de associação com a classe Concern possua multiplicidade [*]. O mesmo sucede com os aspectos não funcionais (NonFunctionalAspect), pois um Viewpoint pode conter nenhum ou vários aspectos que lhe sejam transversais. Desse modo, surge a relação de associação *isCrosscuttedBy* com multiplicidade [*].

Na técnica AORE [22], um concern representa um requisito não funcional do sistema, portanto, e tendo como base a figura 3.16 é possível observar os relacionamentos que cada um destes elementos possui em relação aos restantes. Um Concern, que para além dos atributos nome e descrição, possui também, à semelhança de um *viewpoint*, pelo menos um requisito, daí a multiplicidade [1..*] da sua associação contains com Requirement.

No que diz respeito ao seu relacionamento com *viewpoints*, um *concern* necessita afectar pelo menos um *viewpoint*, caso contrário a sua existência no problema seria desnecessária. Dessa forma, cada elemento Concern contém uma relação *affects* com multiplicidade [1..*] com Viewpoint. Cada relação entre um Viewpoint e Concern é caracterizada pela necessidade de atribuir um peso, demonstrando o grau de importância que um concern tem na perspectiva do *viewpoint* para a funcionalidade em questão. Assim surge a inclusão de uma classe de associação Weight, expressa através do atributo *weight* do tipo Real, que tomará os valores reais entre [0..1].

Um caso particular de um *concern*, na técnica AORE [22], é quando este passa a ser denominado por aspecto não funcional. Ou seja, sempre que um Concern afecte mais que um Viewpoint, é particularizado em NonFunctionalAspect, em que este por sua vez,

obrigatoriamente se relaciona com a classe Viewpoint através da associação crosscuttsVP com multiplicidade [2..*].

Por último, como cada concern pode ter uma contribuição positiva, negativa ou nula com qualquer outro concern, no diagrama da figura encontra-se então também representada a relação entre dois elementos Concern, com multiplicidade [1..*], designada por contributes, que irá conter um tipo (ContributionType) proveniente de Contribution.

De realçar, que um requisito (Requirement) possa ser distinguido entre funcional ou não funcional, dependendo da sua origem (Viewpoint ou Concern respectivamente). Esta condição é garantida através de um atributo booleano isFunctional, que será restringido por intermédio de OCL, mais à frente neste capítulo.

3.3.2. Meta-modelo AORE baseado em viewpoints "modificado"

O modelo apresentado na figura 3.16 contempla apenas a técnica AORE baseada em *viewpoints* utilizada na abordagem de integração desenvolvida para este trabalho. Existe ainda a necessidade de identificar os requisitos aspectuais funcionais de um problema e garantir a sua representação no modelo Problem Frames [13]. Também a figuração de cada aspecto não funcional deve ser aplicada através de diagramas de problema. Torna-se, portanto, indispensável estender o meta-modelo da figura 3.16 com estes conceitos. O modelo de diagrama de classes UML ilustrado na figura 3.17 apresenta, portanto, as relações entre os novos elementos e a sua interacção com os anteriores.

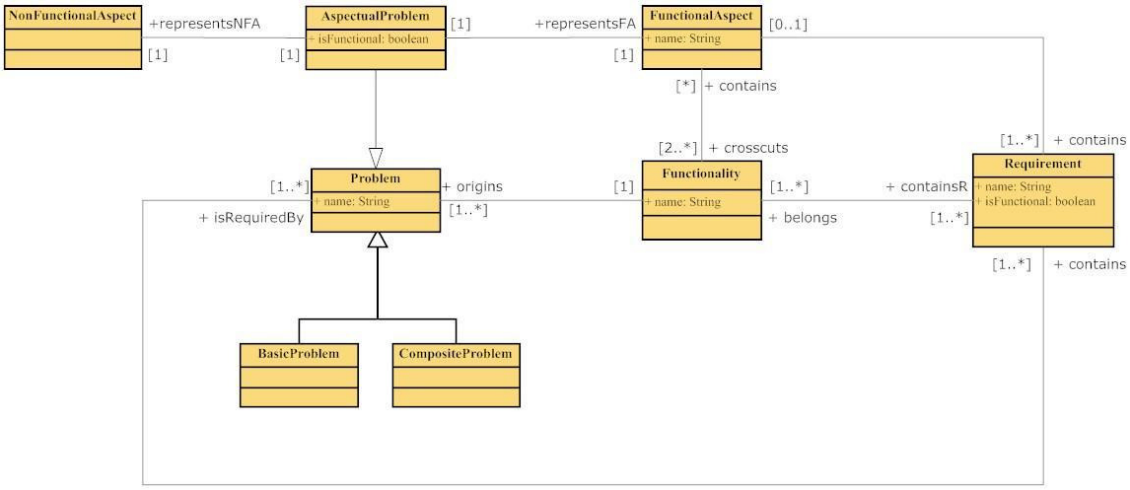


Figura 3.17: Modelo de integração entre AORE e Problem Frames**.**

Neste novo modelo da figura 3.17 estão já presentes os conceitos dos elementos aspectuais funcionais, como apresentado na secção 3.2.3.3, bem como a sua relação com os elementos representativos do modelo de Problem Frames (diagramas de problema).

As diversas funcionalidades que o sistema deve contemplar são expressas pela classe Functionality. Como cada funcionalidade necessita no mínimo de um requisito funcional, então uma classe Functionality deve conter pelo menos um ([1..*]) requisito (Requirement). Nos casos em que um dos seus requisitos seja considerado transversal a outras funcionalidades, essa mesma funcionalidade irá possuir um aspecto funcional (FunctionalAspect). Portanto, cada requisito pode ou não pertencer ([0..1]) a um aspecto funcional, estando dependente da sua própria natureza *crosscutting*. Cada aspecto funcional (FuntionalAspect) detectado tem necessariamente de ser transversal a duas ou mais ([2..*]) funcionalidades (Functionality). Pode no entanto, contemplar mais do que um requisito, já que é utilizada a noção de semelhança entre eles [23]. Estas relações foram observadas nas matrizes de relacionamento da secção 3.2.3.

Introduzindo também os conceitos de representação de requisitos (funcionais ou não funcionais) em problemas no âmbito de Problema Frames [13], a classe Problem foi inicialmente apresentada no meta-modelo de [18]. Como uma funcionalidade é caracterizada através de um ou mais problemas ([1..*]), então também deve existir uma relação de associação entre as classes Problem e Functionality. Cada problema representado na forma de diagrama contém pelo menos um requisito, daí que Problem esteja também relacionado com Requirement. Um problema, para além de representar uma funcionalidade, pode ser igualmente particularizado em problemas básicos (BasicProblem) e compostos (CompositeProblem), como mencionado em [13, 18]. Estendendo essa noção, direccionando-a para aspectos, um problema vai agora permitir representar um aspecto funcional ou não funcional (AspectualProblem). Dessa forma, um AspectualProblem é a representação directa em Problem Frames de um aspecto. Por conseguinte possui uma relação bidireccional com NonFunctionalAspect e FunctionalAspect com multiplicidade 1.

3.3.3. Meta-modelo de Problem Frames integrado com AORE

Os modelos até agora apresentados nas figuras 3.16 e 3.17 serão ainda completados com o modelo desenvolvido em [11], para assim providenciar uma método completo de identificação e especificação de aspectos funcionais e não funcionais recorrendo a Problem Frames (figura 3.18).

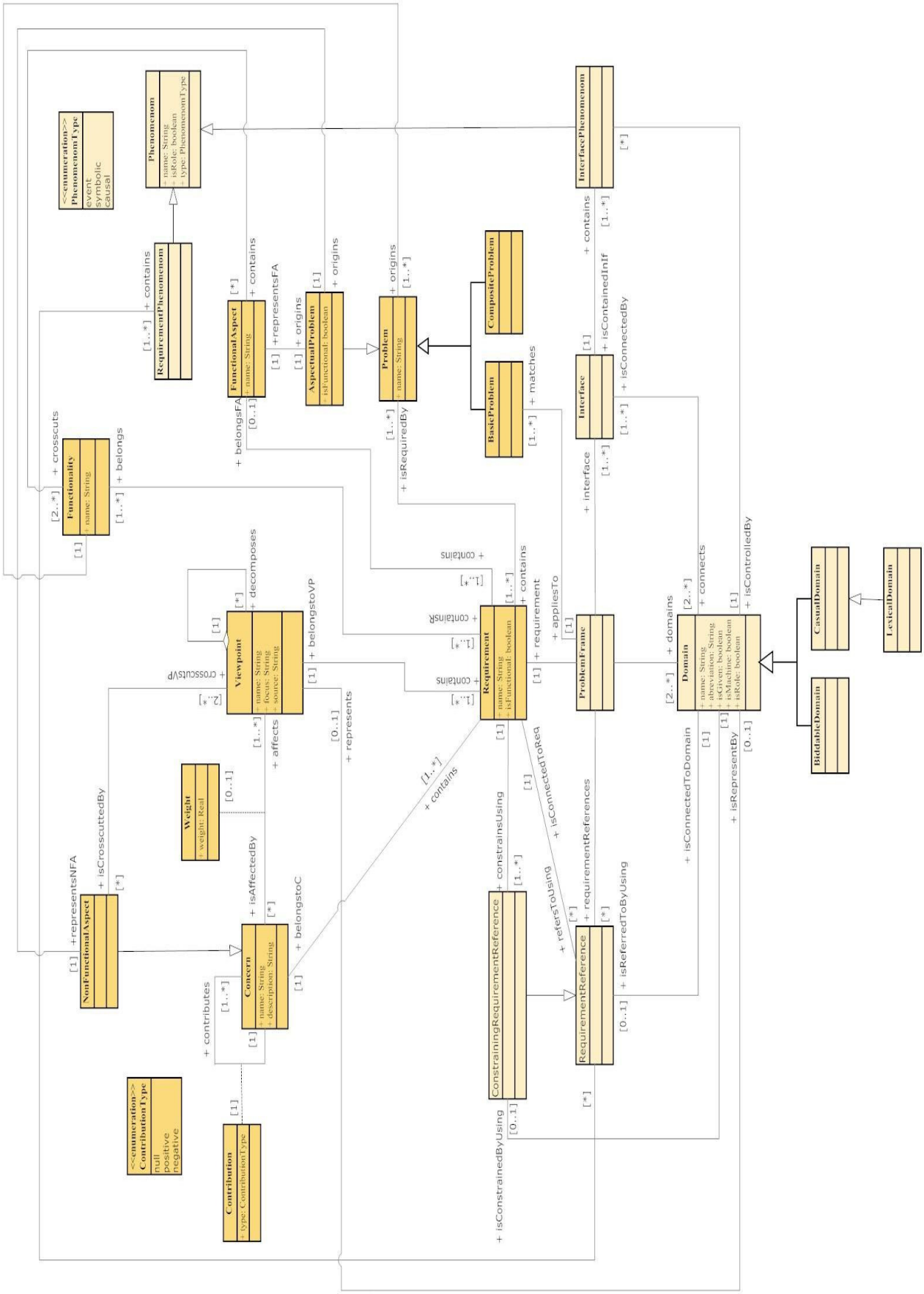


Figura 3.18: Meta-modelo de Problem Frames integrado com AORE**.**

O diagrama de classes UML da figura 3.18 reúne todos os modelos até aqui discutidos (classes de cor laranja), incluindo agora também o modelo de [1] (classes de cor bege).

Para que a interacção entre todos eles esteja completa, falta portanto descrever os relacionamentos existentes. Na figura 3.18 as relações de associação que estabelecem este elo de ligação correspondem, tal como demonstrado na abordagem proposta neste capítulo, são respeitantes às classes Viewpoint e Domain ou entre BasicProblem e ProblemFrame. Como cada *viewpoint* identificado na técnica AORE pode ser representado por um domínio no contexto de Problem Frames [13], aplica-se a relação isRepresentedBy com multiplicidade [0..1]. No sentido inverso, nem sempre um domínio é a representação física de um *viewpoint*. É nesse sentido que um Domain pode representar ou não um *viewpoint*.

Quanto aos problemas básicos, estes aplicam-se directamente a um *problem frame*, sendo então necessário estabelecer uma relação entre Basic Problem e Problem Frame (appliesTo com multiplicidade 1). Um Problem Frame, por sua vez, pode ser aplicado a mais do que um problema básico.

Os problemas aspectuais requerem que um domínio ou fenómeno possa, para além das suas características apresentadas em [11], ser especificado como um padrão para posterior instanciação, daí que origine a criação de um atributo booleano *isRole* nas classes Domain e Phenomenon.

3.3.3.1. Restrições OCL

O modelo final da figura 3.18 vai permitir ainda definir condições e restrições que garantam a integridade das relações, através da descrição de expressões da linguagem formal OCL, dando assim continuidade ao trabalho elaborado em [11]. O contexto em que é definida cada expressão em OCL é identificado por *context* e diz respeito ao elemento do modelo a que ela se aplica, neste caso, as restrições pertencem ao contexto das classes ilustradas nas figuras 3.16 a 3.18. Cada restrição é designada por *invariant*, significando que tem de ser obrigatoriamente cumprida, sendo identificada por *inv*. As condições de integridade abaixo descritas foram verificadas utilizando Papyrus UML [21] e recorrendo também a OCL 2.0 Specification [20]. De seguida são apresentadas as restrições OCL aplicadas ao contexto de cada domínio do meta-modelo (figuras 3.19 – 3.29).

Context Viewpoint

Um requisito de um *viewpoint* é sempre um requisito funcional.

```
inv: self.contains ->forAll(r : Requirement | r.isFuntional)
```

O domínio que represente um *viewpoint* não é um domínio máquina.

```
inv: self.isRepresentedBy->forAll(d : Domain | not d.isMachine)
```

Um *viewpoint* afectado por dois *concerns*, que tenham uma contribuição negativa entre si, vai ter um peso associado.

```
inv: self.isAffectedBy->exists(c1 : Concern |
        c1.contributes->exists(c2 : Concern |
        c1.contribution[contributes]->select(ct : ContributionType |
        ct.contributes= c2 and ct.type=ContributionType::negative)->notEmpty()) ) implies
        self.weight->notEmpty()
```

O nome de um *viewpoint* deve ser único.

```
inv: Viewpoint.allInstances()->forAll(vp1, vp2 : Viewpoint |
        vp1<> vp2 implies vp1.name <> vp2.name)
```

Figura 3.19: Restrições OCL no contexto de Viewpoint**.**

Context Concern

Um requisito de um concern é sempre um requisito não funcional.

```
inv: self.contains->forAll(r : Requirement | not r.isFuntional)
```

Um concern contribui apenas com outros *concerns*, nunca com ele próprio.

```
inv: self.contributes->forAll( c : Concern | c <> self)
```

O nome de um concern deve ser único.

```
inv: Concern.allInstances()->forAll(c1, c2 : Concern | c1<>c2 implies c1.name <> c2.name)
```

Figura 3.20: Restrições OCL no contexto de Concern**.**

Context Weight

Um peso deve compreender um valor real no intervalo [0..1].

```
inv: self.weight>=0.0 and self.weight<=1.0
```

Figura 3.21: Restrições OCL no contexto de Weight**.**

Context Requirement

Um requisito funcional implica que este pertence a um *viewpoint*.

inv: **self**.isFuntional **implies** (**self**.belongstoVP->notEmpty() **and self**.belongstoC->isEmpty())

Um requisito não funcional implica que este pertence a um concern.

Inv**:** **not self**.isFuntional **implies** (**self**.belongstoVP->isEmpty() **and self**.belongstoC->notEmpty())

Um requisito transversal a pelo menos duas funcionalidades pertence a um aspecto funcional.

Inv**: self**.belongs->size()>1 **implies self**.belongsFA->size()=1

Figura 3.22: Restrições OCL no contexto de Requirement**.**

Context Domain

Um domínio máquina implica que este não represente um *viewpoint*.

inv**: self**.isMachine **implies self**.represents->isEmpty()

Figura 3.23: Restrições OCL no contexto de Domain**.**

Context FunctionalAspect

Um requisito de um aspecto funcional é sempre um requisito funcional.

inv: **self**.contains->forAll(r : Requirement | r.isFuntional)

As funcionalidades em que um aspecto funcional lhes é transversal contêm sempre os seus requisitos.

inv: **self**.contains->forAll(r : Requirement | **self**.crosscuts-> forAll(f : Functionality |

f.containsR->includes(r)))

A representação de um aspecto funcional através de um problema é sempre um problema aspectual funcional.

inv: **self**.origins->forAll(ap : AspectualProblem | ap.isFunctional)

O nome de um aspecto funcional deve ser único.

inv: FunctionalAspect.allInstances()->forAll(fa1, fa2 : FunctionalAspect |

fa1<>fa2 **implies** fa1.name <> fa2.name)

Figura 3.24: Restrições OCL no contexto de FunctionalAspect**.**

Context NonFunctionalAspect

A representação de um aspecto não funcional através de um problema é sempre um problema aspectual funcional.

inv: **self**.origins->forAll(ap : AspectualProblem | **not** ap.isFunctional)

Figura 3.25: Restrições OCL no contexto de NonFunctionalAspect**.**

Context Problem

O nome de um problema deve ser único.

inv: Problem.allInstances()->forAll(p1, p2 : Problem | p1<>p2 **implies** p1.name <> p2.name)

Figura 3.26: Restrições OCL no contexto de Problem**.**

Context AspectualProblem

Um problema aspectual funcional implica que a sua origem é proveniente de um aspecto funcional.

inv: **self**.isFunctional **implies** (**self**.representsFA->notEmpty() **and**

self.representsNFA ->isEmpty())

Um problema aspectual não funcional implica que a sua origem é proveniente de um aspecto não funcional.

inv: **not self**.isFunctional **implies** (**self**.representsFA->isEmpty() **and**

self.representsNFA->notEmpty())

Figura 3.27: Restrições OCL no contexto de AspectualProblem**.**

Context BasicProblem

Um problema básico contém um único requisito.

inv: **self**.contains->size()=1

Figura 3.28: Restrições OCL no contexto de BasicProblem**.**

Context Functionality

Cada requisito de uma funcionalidade pertence a pelo menos um problema por ela originado.

inv: **self**.origins->forAll(p : Problem | **self**.containsR->exists(r : Requirement |

p.contains->includes(r)))

Figura 3.29: Restrições OCL no contexto de Functionality**.**

3.4. Resumo

Neste capítulo foi demonstrado o processo de Problem Frames [13] integrado com aspectos, por intermédio do recurso à técnica AORE [22], bem como a utilização de um exemplo simples, de forma a compreender o seu método de aplicação a um problema de software.

Foi também apresentado um meta-modelo formal da abordagem AORE [22] com algumas adaptações para a sua integração com o modelo de Problem Frames introduzido em [11].

No próximo capítulo será apresentado um outro caso de estudo mais complexo, de modo a verificar a aplicabilidade da abordagem em que se centra esta dissertação, e um comparativo com outras abordagens semelhantes.

4. Caso de estudo e comparação com outras abordagens

Neste capítulo vai ser apresentado um outro caso de estudo, nomeadamente o sistema Health Watcher, em que será aplicada a abordagem integrada entre AORE e Problem Frames, para a sua validação. Será também realizada uma comparação entre com outras abordagens, nomeadamente, orientadas a aspectos e que integrem aspectos em Problem Frames.

4.1. Caso de estudo – Health Watcher

O Health Watcher é descrito em [25] e consiste num exemplo real de um sistema desenvolvido para melhorar a qualidade dos serviços providenciados por instituições prestadoras de cuidados de saúde.

Este documento especifica os requisitos para o sistema público de saúde designado por HEALTH-WATCHER, que fornece aos engenheiros, a informação necessária para o desenvolvimento do sistema. O propósito deste sistema é receber e controlar as reclamações e notificações, providenciando também importantes informações aos utentes acerca do sistema de saúde. Com a implementação do sistema HEALTH-WATCHER, o sistema de saúde público melhorará consideravelmente:

• O controlo de reclamações (denúncias e notificações).

• A qualidade do serviço em espalhar a sua informação.

O cidadão será capaz de aceder ao sistema, solicitar informação sobre serviços de saúde ou fazer as suas reclamações. Exemplos de consultas possíveis que um cidadão pode efectuar são, por exemplo: informação sobre doenças (descrição, sintomas, prevenção da doença), campanhas de vacinação, informação sobre reclamações realizadas pelo cidadão (nome e dados da pessoa, data, descrição e observações da reclamação, situação (aberta, suspensa, fechada), análise técnica, data e funcionário responsável pela análise), unidades de saúde responsáveis por uma determinada especialidade, especialidades de uma unidade de saúde.

No evento de uma reclamação, esta ficará registada no sistema e será encaminhada para um departamento específico (representado por um assistente registado), que será capaz de realizar o procedimento e retornar uma resposta quando a análise estiver terminada. Esta solução ficará registada no sistema, ficando disponível para consultas. Os diversos tipos de reclamação são:

- Reclamação Animal: apreensão de animais, controlo de pragas (roedores, escorpiões, morcegos, etc.), doenças relacionadas com mosquitos, mau trato de animais. Informação adicional requerida: tipo de animal, quantidade de animais, local e data do distúrbio.
- Reclamação Alimentar: casos em que se suspeite de ingestão de comida infectada. Informação adicional requerida: nome e dados da vítima, número de pessoas que tenham ingerido a comida, número de pessoas doentes; número de pessoas que tenham sido enviadas para um hospital e número de pessoas falecidas, localização onde os pacientes foram tratados e alimentos suspeitos.
- Reclamação Diversa: casos relacionados com diferentes razões que não são acima mencionados (restaurantes com problemas de higiene, vazamento de esgotos, camiões de transporte de aguas suspeitos, etc.).

O produto também será colocado para utilização pública em quiosques em diversos pontos estratégicos, em que o cidadão poderá por si próprio efectuar as suas reclamações e pedidos de informação. Também, o novo sistema deve permitir troca de informação com o sistema SVS (Sistema de Vigilância Sanitária). Finalmente, um relatório com estatísticas apresentando a frequência dos tipos de reclamações é enviado directamente para o director de cada unidade de saúde, todos os meses.

De seguida será aplicada a abordagem de Problem Frames integrada com AORE.

4.1.1. Identificação e especificação de requisitos

4.1.1.1. Identificação de viewpoints e especificação de requisitos

Ao analisar a descrição do sistema, os *viewpoints* identificados neste caso de estudo são: Cidadão, Assistente Registado, Administrador de Unidade de Saúde, Quiosque, Sistema de Vigilância Sanitária, Administrador do Sistema, Director da Unidade de Saúde (tabelas 4.1 – 4.7).

Tabela 4.1: *Viewpoint* Cidadão**.**

Atributos do viewpoint	Descrição do atributo
Nome:	Cidadão.
Foco:	Aceder ao sistema Health-Watcher para solicitar informações sobre serviços de saúde e efectuar reclamações.
Concerns:	Tempo de resposta, Multi-acesso, Segurança.
Fontes:	Interface do Quiosque, resposta à reclamação, informação sobre serviços de saúde.
Requisitos:	R1 - O Cidadão acede ao sistema. R1.1 - O Cidadão efectua o login no sistema. R1.2 - O Cidadão efectua logout do sistema. R2 - O Cidadão solicita informação sobre serviços de saúde ao sistema. R2.1 - O Cidadão solicita informação sobre doenças (descrição, sintomas, prevenção da doença). R2.2 - O Cidadão solicita informação sobre campanhas de vacinação. R2.3 - O Cidadão solicita informação sobre reclamações (nome e dados do autor da reclamação, data, descrição e observações, situação (aberta, suspensa, fechada), análise técnica, data e funcionário). R2.4 - O Cidadão solicita informação sobre que unidade de saúde é responsável por uma especialidade específica. R2.5 - O Cidadão solicita informação sobre as especialidades de uma unidade de saúde. R3 - Se tiver feito login, o Cidadão regista a reclamação no sistema, de acordo com o seu tipo (animal, alimentar, diverso).

Tabela 4.2: *Viewpoint* Assistente Registado**.**

Atributos do viewpoint	Descrição do atributo
Nome:	Assistente Registado.
Foco:	Representa um departamento específico e é responsável por dar seguimento às reclamações, devendo retornar uma resposta quando a sua análise estiver terminada.
Concerns:	Tempo de resposta, Multi-acesso, Segurança.
Fontes:	Reclamações do Cidadão, informação sobre animais, alimentos e diversos recursos.
Requisitos:	R1 - O Assistente Registado acede ao sistema. R1.1 - O Assistente Registado efectua o login no sistema. R1.2 - O Assistente Registado efectua logout do sistema. R2 - O Assistente Registado solicita uma reclamação do sistema, dependendo do seu tipo. R3 - O Assistente Registado analisa e actualiza o estado da reclamação (aberta, suspensa, fechada) no sistema. R4 - O Assistente Registado retorna uma resposta para a caixa de correio electrónico do Cidadão quando a análise da reclamação estiver terminada.

Tabela 4.3: *Viewpoint* Administrador da Unidade de Saúde**.**

Atributos do viewpoint	Descrição do atributo
Nome:	Administrador da Unidade de Saúde (AUS).
Foco:	O AUS é responsável pela administração e manutenção da informação pertencente à sua unidade de saúde.
Concerns:	Tempo de resposta, Multi-acesso, Segurança.
Fontes:	Informações sobre especialidades de unidades de saúde, doenças, campanhas de vacinação, estatísticas sobre reclamações.
Requisitos:	R1 - O AUS acede ao sistema. R1.1 - O AUS efectua o login no sistema. R1.2 - O AUS efectua logout do sistema. R2 - O AUS insere informação sobre unidades de saúde. R2.1 - O AUS insere as especialidades de uma unidade de saúde. R2.2 - O AUS insere informação acerca de doenças (descrição, sintomas, prevenção). R2.3 - O AUS insere informação sobre campanhas de vacinação. R3 - O AUS remove informação sobre unidades de saúde. R3.1 - O AUS remove especialidades de uma unidade de saúde. R3.2 - O AUS remove informação acerca de doenças (descrição, sintomas, prevenção). R3.3 - O AUS remove informação sobre campanhas de vacinação. R4 - O AUS actualiza informação sobre unidades de saúde. R4.1 - O AUS actualiza as especialidades de uma unidade de saúde. R4.2 - O AUS actualiza informação acerca de doenças (descrição, sintomas, prevenção). R4.3 - O AUS actualiza informação sobre campanhas de vacinação.

Tabela 4.5: *Viewpoint* Sistema de Vigilância Sanitária**.**

Atributos do viewpoint	Descrição do atributo
Nome:	Sistema de Vigilância Sanitária (SVS).
Foco:	SVS troca informação com o sistema Health-Watcher.
Concerns:	Tempo de resposta, Multi-acesso.
Fontes:	Informação sobre doenças, especialidades de unidades de saúde, campanhas de vacinação.
Requisitos:	R1 - O SVS envia informação para o sistema. R1.1 - O SVS envia informação sobre doenças para o sistema. R1.2 - O SVS envia informação sobre unidades de saúde para o sistema. R1.3 - O SVS envia informação sobre campanhas de vacinação para o sistema. R2 - O SVS recebe informação do sistema. R2.1 - O SVS recebe informação sobre doenças do sistema. R2.2 - O SVS recebe informação sobre unidades de saúde do sistema. R2.3 - O SVS recebe informação sobre campanhas de vacinação do sistema.

Tabela 4.6: *Viewpoint* Administrador de sistema**.**

Atributos do viewpoint	Descrição do atributo
Nome:	Administrador do Sistema (AS).
Foco:	O AS é responsável pela manutenção do sistema Health-Watcher.
Concerns:	Tempo de resposta, Segurança.
Fontes:	Informação sobre unidades de saúde, tipos de reclamação
Requisitos:	R1 - O AS acede ao sistema. R1.1 - O AS efectua o login no sistema. R1.2 - O AS efectua logout do sistema. R2 - O AS gere a informação de unidades de saúde no sistema Health-Watcher. R2.1 - O AS insere uma nova unidade de saúde no sistema. R2.2 - O AS remove uma unidade de saúde do sistema. R2.3 - O AS actualiza uma unidade de saúde no sistema. R3 - O AS gere a informação de reclamações no sistema Health-Watcher. R3.1 - O AS insere um novo tipo de reclamação no sistema. R3.2 - O AS remove um tipo de reclamação do sistema. R3.3 - O AS actualiza um tipo de reclamação no sistema.

Tabela 4.7: *Viewpoint* Director da Unidade de Saúde**.**

Atributos do viewpoint	Descrição do atributo
Nome:	Director da Unidade de Saúde (DUS).
Foco:	O DUS recebe um relatório com as estatísticas da frequência dos tipos de reclamação.
Concerns:	Tempo de resposta, Multi-acesso, Segurança.
Fontes:	Relatórios, tipos de reclamação
Requisitos:	R1 - O DUS acede ao sistema. R1.1 - O DUS efectua o login no sistema. R1.2 - O DUS efectua logout do sistema. R2 - O DUS recebe um relatório mensal com estatísticas sobre reclamações efectuadas.

4.1.1.2. Identificação e especificação de concerns

Os *concerns* considerados neste caso de estudo são os seguintes: Tempo de resposta, Multi-acesso e Segurança (tabelas 4.8 – 4.10).

Tabela 4.8: Concern Tempo de resposta**.**

Atributos do concern	Descrição do atributo
Nome:	Tempo de resposta (TR).
Descrição:	Permite ao sistema reagir num curto período de tempo aos acessos a dados e execução de operações, através da indexação das bases de dados.
Viewpoints influenciados:	Cidadão, Quiosque, SVS.
Requisitos:	R1 - O sistema tem que ter TR para processar os pedidos do Cidadão (acesso, informação, reclamação). R2 - O sistema tem que ter TR para trocar informações com o SVS. R3 - O sistema tem que ter TR para processar os pedidos do Cidadão (acesso, informação, reclamação), através do Quiosque. R4 - O sistema tem que ter TR para processar os pedidos do Assistente Registado. R5 - O sistema tem que ter TR para processar os pedidos do AUS. R6 - O sistema tem que ter TR para processar os pedidos do AS. R7 - O sistema tem que ter TR para processar os pedidos do DUS.

Tabela 4.9: Concern Multi-acesso**.**

Atributos do concern	Descrição do atributo
Nome:	Multi-acesso.
Descrição:	O sistema deve suportar vários pedidos em simultâneo de diferentes fontes, atribuindo acesso de forma concorrente.
Viewpoints influenciados:	Cidadão, Assistente Registado, AUS, Quiosque, SVS.
Requisitos:	R1 - Receber múltiplos pedidos por parte do Cidadão. R2 - Aceitar diversos pedidos do Assistente Registado para a análise, actualização e resposta das reclamações. R3 - Permitir a manutenção da informação sobre unidades de saúde (especialidades, doenças, campanhas de vacinação) por vários AUS. R4 - Receber e responder a múltiplos pedidos por parte do Cidadão, através do Quiosque. R5 - Trocar diversos tipos de informação com o SVS simultaneamente. R6 - Emitir diversos relatórios para diversos DUS.

Tabela 4.10: Concern Segurança**.**

Atributos do concern	Descrição do atributo
Nome:	Segurança.
Descrição:	O sistema deve providenciar segurança no acesso apenas a pessoas autorizadas, através de uma palavra-chave.
Viewpoints influenciados:	Cidadão, Assistente Registado, AUS, Quiosque, Administrador do sistema, DUS.
Requisitos:	R1 - Controlar o acesso ao Cidadão quando este tenta efectuar login no sistema. R2 - Controlar o acesso ao Assistente Registado quando este tenta efectuar login no sistema. R3 - Controlar o acesso ao AUS quando este tenta efectuar login no sistema. R4 - Controlar o acesso ao Cidadão quando este tenta efectuar login no sistema, através do Quiosque. R5 - Controlar o acesso ao Administrador do sistema quando este tenta efectuar login. R6 - Controlar o acesso ao DUS quando este tenta efectuar login no sistema.

4.1.2. Identificação de aspectos não funcionais

4.1.2.1. Relacionar viewpoints e concerns

Nesta secção é realizado o relacionamento entre os *viewpoints* identificados e *concerns* considerados no sistema Health Watcher (figura 4.11).

Tabela 4.11: Matriz de relacionamento entre *concerns* e *viewpoints***.**

Viewpoints / Concerns	Cidadão	AR	AUS	Quiosque	SVS	AS	DUS
Tempo de resposta	X	X	X	X	X	X	X
Multi-acesso	X	X	X	X	X		X
Segurança	X	X	X	X		X	X

(AR: Assistente registado**; AUS:** Administrador Unidade Saúde**; SVS:** Sistema Vigilância Sanitária**; AS:** Administrador do sistema; **DUS:** Director Unidade Saúde**)**

4.1.2.2. Identificar crosscutting concerns

Os *crosscutting concerns* identificados são: Tempo de resposta, Multi-acesso e Segurança. Normalmente, e tal como acontece neste caso de estudo, a maioria dos requisitos não funcionais afecta e influencia mais do que um *viewpoint*.

4.1.2.3. Identificação e resolução de conflitos

As tabelas 4.12 e 4.13 apresentam as matrizes de contribuições entre aspectos e atribuição de prioridades entre aspectos conflituosos, respectivamente.

Tabela 4.12: Matriz de contribuições entre aspectos**.**

Aspectos / Aspectos	Tempo de resposta	Multi-acesso	Segurança
Tempo de resposta		+	-
Multi-acesso	-		
Segurança	-		

Tabela 4.13: Matriz de prioridades em *viewpoints* entre aspectos conflituosos**.**

Viewpoints / Aspectos	Cidadão	AR	AUS	Quiosque	SVS	AS	DUS
Tempo de resposta	0.8	0.9	0.9	0.8	0.8	0.9	0.9
Multi-acesso	0.6	0.7	0.7	0.6	0.5		0.7
Segurança	0.5	0.8	0.8	0.5		0.8	0.8

(AR: Assistente registado**; AUS:** Administrador Unidade Saúde**; SVS:** Sistema Vigilância Sanitária**; AS:** Administrador do sistema; **DUS:** Director Unidade Saúde**)**

4.1.3. Identificação de aspectos funcionais

4.1.3.1. Identificação de funcionalidades e requisitos

As funcionalidades identificadas no caso de estudo do Health Watcher são as seguintes: Registar reclamação, Solicitar informação, Administrar informação, Administrar sistema, Trocar informação, Emitir relatório e Analisar reclamação.

4.1.3.2. Relacionar viewpoints, requisitos e funcionalidades

A tabela 4.14 demonstra a matriz com os relacionamentos entre os *viewpoints* e seus requisitos com as funcionalidades do sistema.

Tabela 4.14: Matriz de relacionamento entre *viewpoints*, requisitos e funcionalidades**.**

Func. / VP	Registar reclamação	Solicitar informação	Administrar Informação	Administrar sistema	Trocar informação	Emitir relatório	Analisar reclamação
	Requisitos de *Viewpoints*						
Cidadão	C.R1.1; C.R1.2; C.R3;	C.R2;					
AR							AR.R1.1; AR.R1.2; AR.R2; AR.R3; AR.R4
AUS			AUS.R1.1; AUS.R1.2; AUS.R2; AUS.R3; AUS.R4				
Quiosque	Q.R1.1; Q.R1.2; Q.R3;	Q.R2; Q.R4					
SVS					SVS.R1; SVS.R2;		
AS				AS.R1.1; AS.R1.2; AS.R2; AS.R3;			
DUS						DUS.R1.1; DUS.R1.2; DUS.R2	

(AR: Assistente registado**; AUS:** Administrador Unidade Saúde**; SVS:** Sistema Vigilância Sanitária**; AS:** Administrador do sistema; **DUS:** Director Unidade Saúde**)**

4.1.3.3. Identificar aspectos funcionais

Como todos os utilizadores têm de efectuar login ou logout no sistema, logo, ambos os requisitos são *crosscutting*, daí que sejam considerados aspectos funcionais. O mesmo sucede com as acções de inserção, remoção ou actualização de dados no sistema. A tabela 4.15 apresenta os aspectos funcionais identificados e os requisitos que o originam. A tabela 4.16 apresenta as funcionalidades compostas apenas pelos requisitos não aspectuais.

Tabela 4.15: Aspectos funcionais**.**

Aspecto funcional	Requisitos
AspLogin	(C.R1.1 – AUS.R1.1 – AS.R1.1 – AR.R1.1 – DUS.R1.1) – Efectuar Login
AspLogout	(C.R1.2 – AUS.R1.2 – AS.R1.2 – AR.R1.2 – DUS.R1.2) – Efectuar Logout
AspInserirInfo	(AUS.R2 – AS.R2.1 – AS.R3.1) – Inserir informação no sistema
AspRemoverInfo	(AUS.R3 – AS.R2.2 – AS.R3.2) – Remover informação no sistema
AspActualizarInfo	(AUS.R4 – AS.R2.3 – AS.R3.3) – Actualizar informação no sistema

Tabela 4.16: Matriz de relacionamento entre funcionalidades e requisitos de *viewpoints* com remoção de aspectos funcionais**.**

Func. / VP	Registar reclamação	Solicitar informação	Administrar informação	Administrar sistema	Trocar informação	Emitir relatório	Analisar reclamação
	Requisitos de *Viewpoints*						
Cidadão	C.R3	C.R2					
AR							AR.R2; AR.R3; AR.R4
AUS							
Quiosque	Q.R1.1; Q.R1.2; Q.R3	Q.R2; Q.R4					
SVS					SVS.R1; SVS.R2;		
AS							
DUS						DUS. R2	

(AR: Assistente registado**; AUS:** Administrador Unidade Saúde**; SVS:** Sistema Vigilância Sanitária**; AS:** Administrador do sistema; **DUS:** Director Unidade Saúde**)**

4.1.4. Problem Frames

4.1.4.1. Diagrama de Contexto

Os domínios do problema Health Watcher e suas relações são ilustrados no diagrama de contexto na figura 4.1.

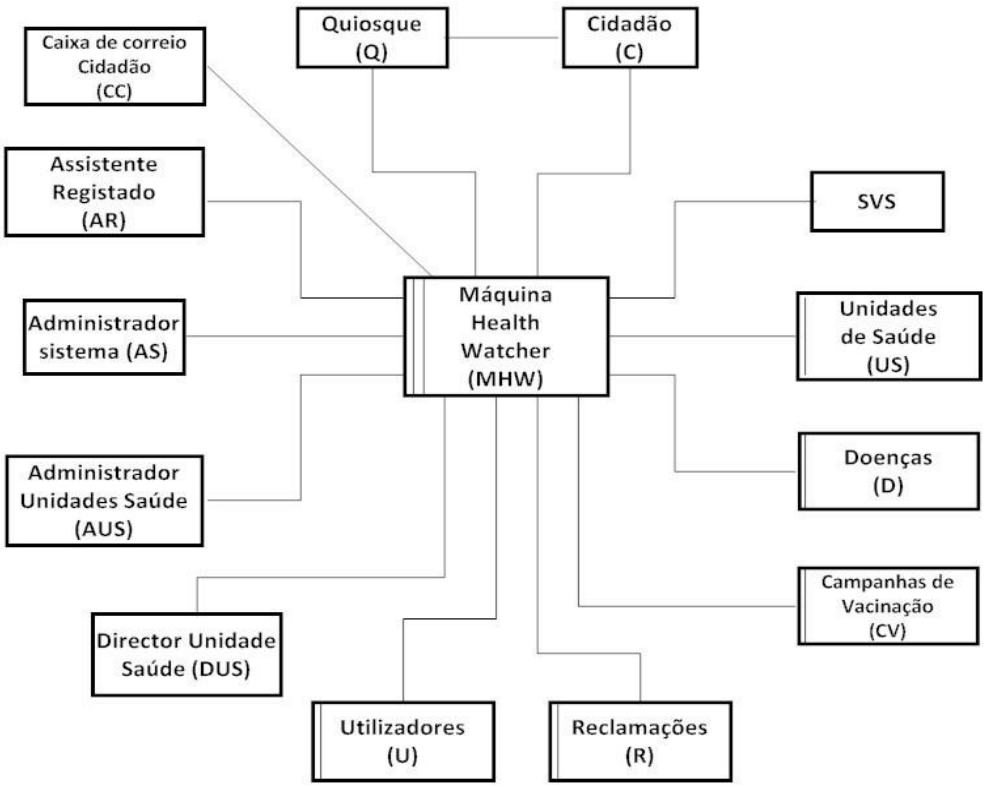
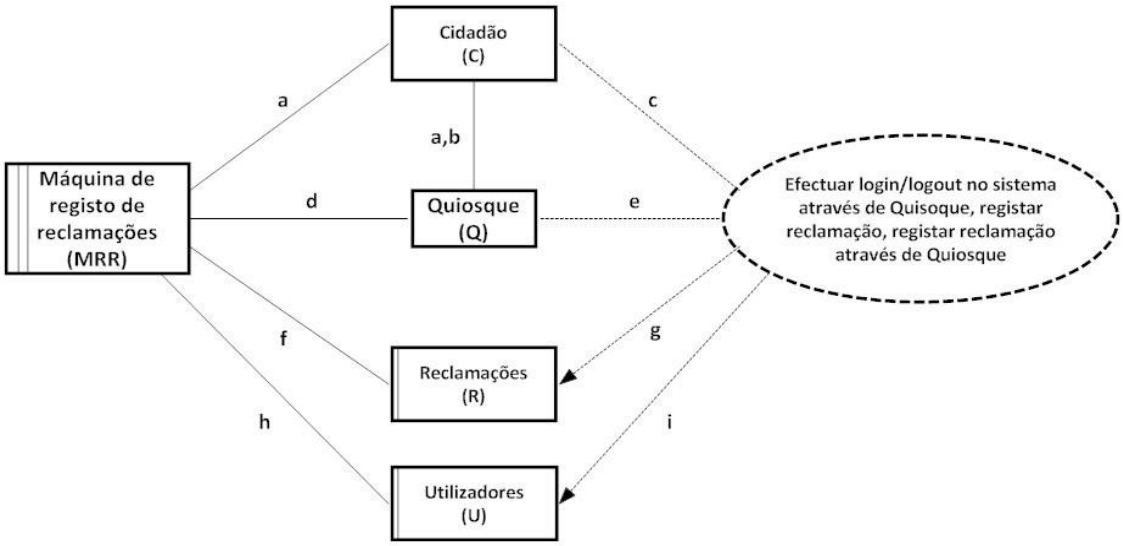


Figura 4.1: Diagrama de Contexto**.**

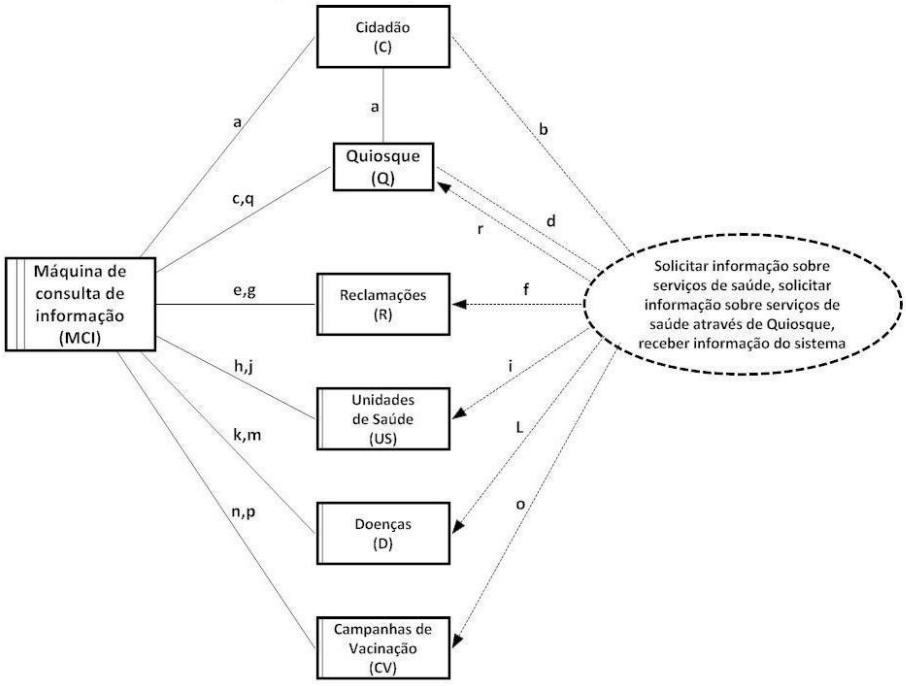
4.1.4.2. Diagramas de problema

Nesta secção os diagramas de problema representam as funcionalidades da tabela 4.16, isto é, não consideram os requisitos aspectuais. Como algumas funcionalidades eram compostas inteiramente por requisitos aspectuais (Administrar informação e Administrar sistema) não podem estar aqui ilustradas. As figuras 4.2 e 4.3 apresentam os diagramas de problema respectivos às funcionalidades de Registar reclamação e Solicitar informação.



a: C!{registarReclamação()}
b: C!{login(), logout()}
c: C!{utilizador, palavra chave, reclamação}
d: Q!{login(), logout(), registarReclamação()}
e: Q!{utilizador, palavra chave, reclamação}
f: MRR!{registarReclamação(reclamação)}
g: R!{Reclamação registada}
h: MRR!{login(utilizador, palavra chave), logout(utilizador)}
i: U!{Login efectuado, Logout efectuado}

Figura 4.2: Diagrama de problema Registar reclamação**.**



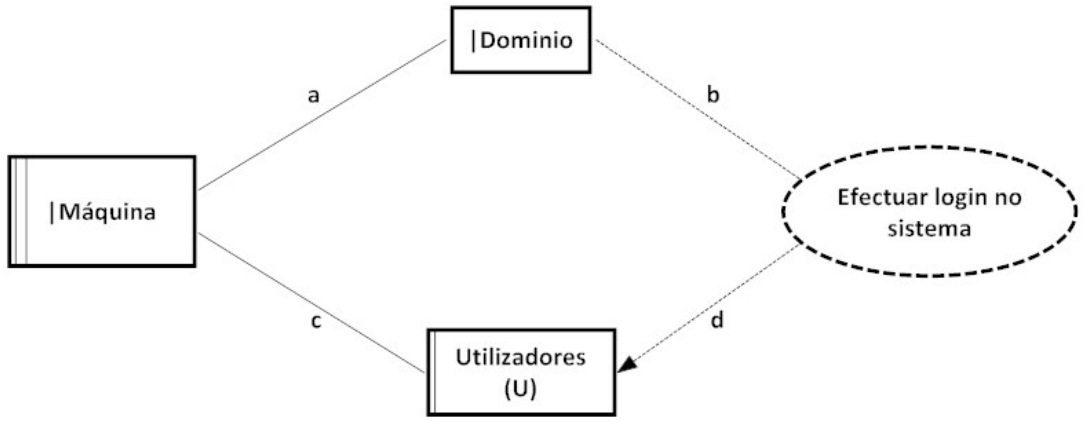
a: C!{informação(tipo)}
b: C!{tipo da informação}
c: Q!{informação(tipo)}
d: Q!{tipo da informação}
e: MCI!{consultarReclamação(reclamação)}
f: R!{reclamação existente}
g: R!{reclamação}
h: MCI!{consultarUS(especialidade), consultarUS(unidadeSaúde)}
i: US!{unidade saúde existente, especialidade existente}
j: US!{unidade saúde, especialidade}
k: MCI!{consultarDoença(doença)}
L: D!{doença existente}
m: D!{doença}
n: MCI!{consultarVacinação(campanha)}
o: CV!{campanha vacinação existente}
p: CV!{campanha vacinação}
q: MCI!{receberInformação(informação)}
r: Q!{informação recebida}

Figura 4.3: Diagrama de problema Solicitar informação**.**

Os restantes diagramas podem ser consultados no Apêndice C.

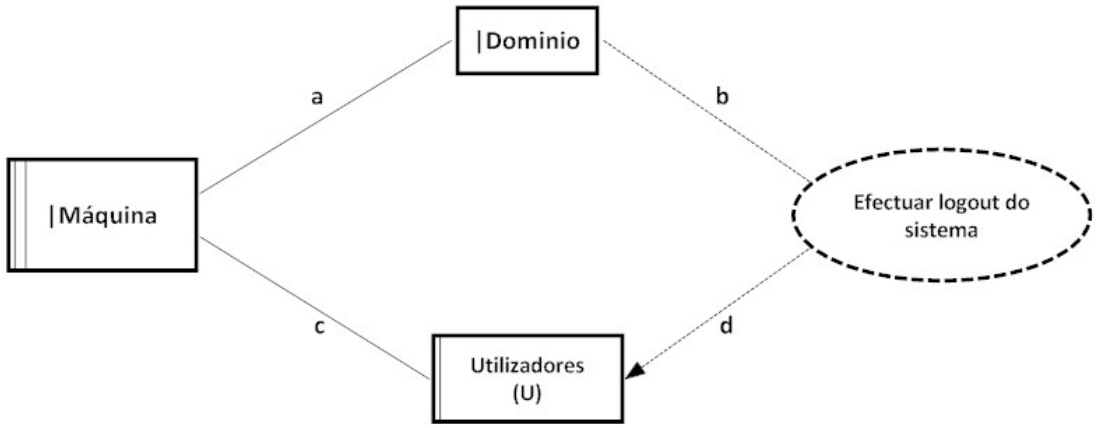
4.1.4.3. Diagramas de problema aspectuais

Os diagramas de problema aspectuais derivam dos aspectos funcionais identificados, e que estão presentes na tabela 4.15. Apenas estarão aqui figurados nas figuras 4.4 e 4.5, os aspectos Login (AspLogin) e Logout (AspLogout) respectivamente.



a: Dominio!{login()}
b: Dominio!{utilizador, palavra chave}
c: MáquinaI!{login(utilizador, palavra chave)}
d: U!{Login efectuado}

Figura 4.4: Diagrama de problema aspectual Login (AspLogin)**.**



a: Dominio!{logout()}
b: Dominio!{utilizador}
c: MáquinaI!{logout(utilizador)}
d: U!{Logout efectuado}

Figura 4.5: Diagrama de problema aspectual Logout (AspLogout)**.**

Os restantes diagramas de problema aspectuais encontram-se no Apêndice C.

4.1.4.4. Operacionalização de aspectos não funcionais

Os aspectos não funcionais do problema Health Watcher considerados na secção 4.1.2 (Tempo de resposta, Multi-acesso e Segurança) são agora alvo de representação através de diagramas de problema (figura 4.6).

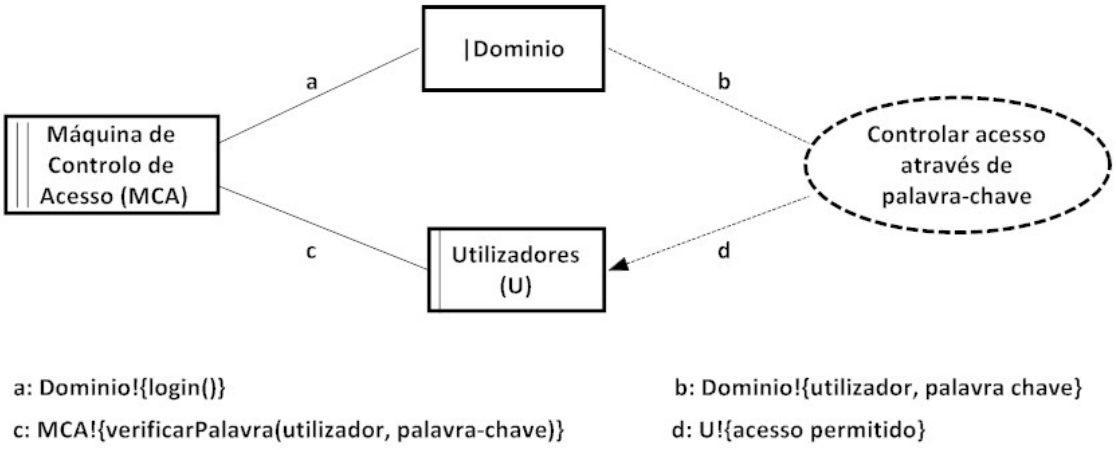
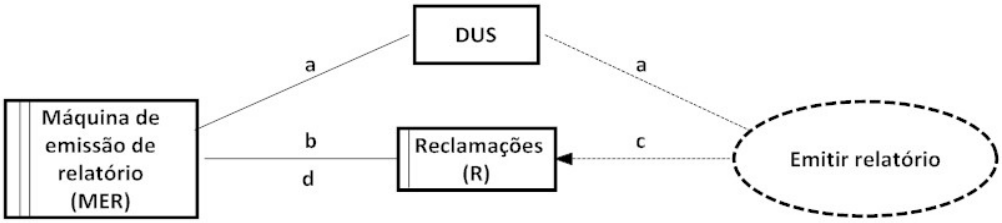


Figura 4.6: Diagrama de problema aspectual não funcional Segurança**.**

Os restantes diagramas de problema aspectuais não funcionais que dizem respeito ao Multi-acesso e Tempo de Resposta estão representados no Apêndice C.

4.1.4.5. Decomposição e especificação de problemas

O problema correspondente à funcionalidade Emitir relatório da tabela 4.17 já se encontra na forma mais simples, pois diz respeito apenas a um requisito. O *problem frame* elementar que lhe corresponde é o *Simple Workpieces*. Este será então ilustrado na figura 4.7, juntamente com o *frame concern* correspondente.



Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o DUS solicita à MER a emissão de relatório através do comando emitirRelatório().	Requisito
2. A MER acede à base de dados Reclamações e efectua o comando emitirRelatório(relatório).	Especificação da máquina
3. Resultando no processamento de estatisticas sobre reclamações e orginando um relatório.	Descrição de domínio
4. Deste modo é alcançado o requisito Emitir relatório através de relatório emitido.	Requisito

Figura 4.7: Diagrama de problema Emitir relatório com *frame concern***.**

Os restantes diagramas de problema decompostos e respectivos *frames concern* encontram-se na secção de Apêndice C.

4.1.5. Regras de composição de aspectos

4.1.5.1. Regras de composição de aspectos funcionais

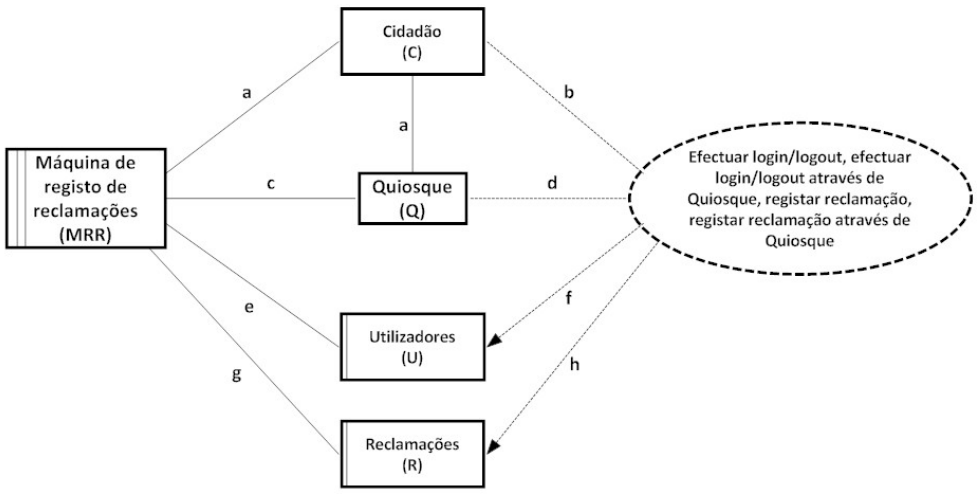
Os aspectos funcionais em que serão aplicadas as regras de composição são Login (AspLogin) e Logout (AspLogout). Estes serão compostos às funcionalidades de Registar reclamação e Emitir relatório da tabela 4.17 (figuras 4.8 – 4.13).

```
Compose aspect AspLogin with Registar reclamação
Bind domain |Máquina to Máquina de registo de reclamações
Bind domain |Dominio to Cidadão
```

Figura 4.8: Instanciação do diagrama de problema aspectual Login a Registar reclamação**.**

```
Compose aspect AspLogout with Registar reclamação
Bind domain |Máquina to Máquina de registo de reclamações
Bind domain |Dominio to Cidadão
```

Figura 4.9: Instanciação do diagrama de problema aspectual Logout a Registar reclamação**.**



a: C!{login(), logout(), registarReclamação()}
b: C!{utilizador, palavra chave, reclamação}
c: Q!{login(), logout(), registarReclamação()}
d: Q!{utilizador, palavra chave, reclamação}
e: MRR!{login(utilizador, palavra chave), logout(utilizador)}
f: U!{Login efectuado, Logout efectuado}
g: MRR!{registarReclamação(reclamação)}
h: R!{reclamação registada}

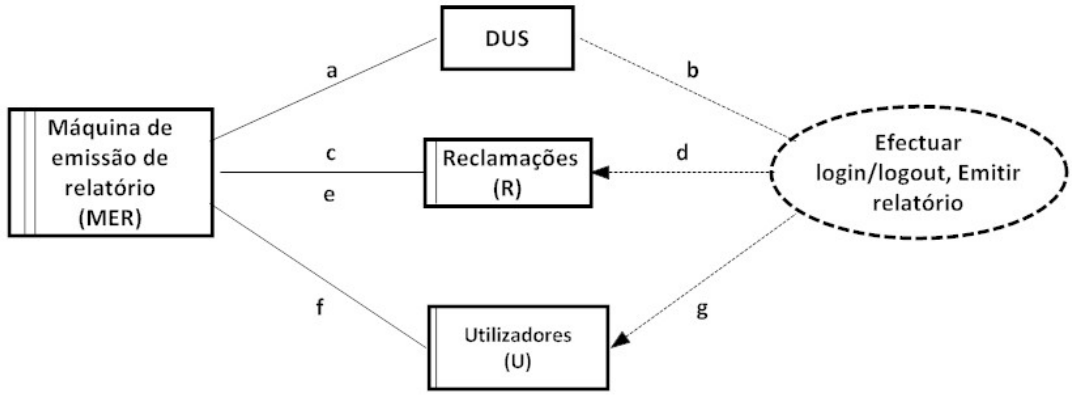
Figura 4.10: Diagrama de problema Registar reclamação composto**.**

```
Compose aspect AspLogin with Emitir relatório
Bind domain |Máquina to Máquina de emissão de relatório
Bind domain |Dominio to Director Unidade de Saúde
```

Figura 4.11: Instanciação do diagrama de problema aspectual Login a Emitir relatório**.**

```
Compose aspect AspLogout with Emitir relatório
Bind domain |Máquina to Máquina de emissão de relatório
Bind domain |Dominio to Director Unidade de Saúde
```

Figura 4.12: Instanciação do diagrama de problema aspectual Logout a Emitir relatório**.**



a: DUS!{login(), logout(), emitirRelatório()}
b: DUS!{utilizador, palavra chave, emitirRelatório()}
c: MER!{emitirRelatório(relatório)}
d: R!{relatório emitido}
e: R!{relatório}
f: MER!{login(utilizador, palavra chave), logout(utilizador)}
g: U!{Login efectuado, Logout efectuado}

Figura 4.13: Diagrama de problema Emitir relatório composto**.**

As restantes regras de composição e diagramas de problema encontram-se na secção de Apêndice C.

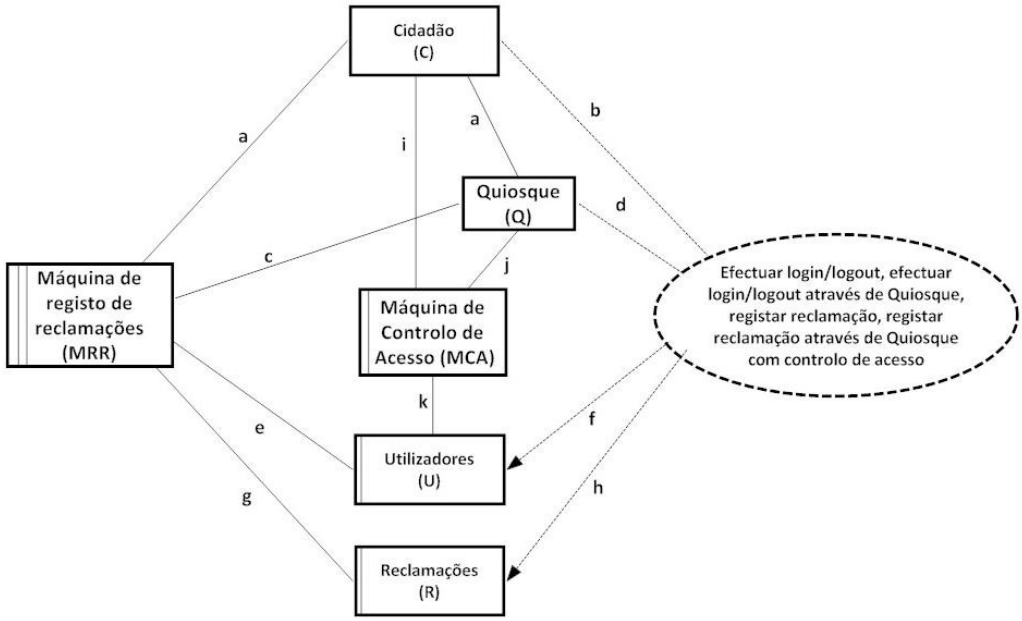
4.1.5.2. Regras de composição de aspectos não funcionais

O aspecto não funcional de Segurança vai ser instanciado ao problema Registar reclamação através das regras de composição de aspectos não funcionais (figuras 4.14 e 4.15).

```
Compose aspect Segurança with Registar Reclamação
Bind domain |Dominio to Cidadão

Compose aspect Segurança with Registar Reclamação
Bind domain |Dominio to Quiosque
```

Figura 4.14: Instanciação do diagrama de problema aspectual Segurança a Registar reclamação**.**



a: C!{login(), logout(), registarReclamação()}
b: C!{utilizador, palavra chave, reclamação}
c: Q!{login(), logout(), registarReclamação()}
d: Q!{utilizador, palavra chave, reclamação}
e: MRR!{login(utilizador, palavra chave), logout(utilizador)}
f: U!{login efectuado, Logout efectuado, acesso permitido}
g: MRR!{registarReclamação(reclamação)}
h: R!{reclamação registada}
i: C!{login()}
j: Q!{login()}
k: MCA!{verificarPalavra(utilizador, palavra-chave)}

Figura 4.15: Diagrama de problema Registar reclamação composto com Segurança.

As restantes regras de composição e diagramas de problema encontram-se na secção de Apêndice C.

4.2. Comparação com outras abordagens

Esta secção vai permitir comparar a abordagem de Problem Frames integrada com AORE apresentada na no capítulo 3, aplicando diversos critérios.

Os critérios seleccionados para efeitos de comparação das abordagens intervenientes são os seguintes:

- "Requisitos funcionais", identifica qual o método utilizado para obter e descrever os requisitos funcionais do sistema;
- "Requisitos não funcionais", indicando se uma abordagem suporta requisitos não funcionais;
- "*Crosscutting concerns*", caso existam métodos de identificação de requisitos *crosscutting*. Vão ser considerados ambos os tipos de requisitos (funcionais e não funcionais);
- "Ferramenta de suporte", considerando as ferramentas que servem de suporte à utilização da abordagem;
- "Resolução de conflitos", dependendo da possibilidade da abordagem lidar com situações conflituosas entre *concerns* ou aspectos não funcionais;
- "Mecanismos de composição", para a existência de algum tipo de mecanismo de composição.

Este critérios foram seleccionados com base na importância que cada um representa, quer nos modelos de EROA, quer nas abordagens que integram aspectos em Problem Frames.

Assegurar uma forma simples e objectiva de estruturar, principalmente de um modo organizado, os requisitos funcionais de um sistema é definitivamente um factor a ter em conta. É importante também considerar a capacidade de reconhecimento de requisitos não funcionais e a detecção de propriedades *crosscutting*, pois desse modo são minimizados os seus efeitos no desenvolvimento e manutenção do software. A possibilidade de utilizar uma ferramenta, que auxilie na estruturação e descrição do sistema que se encontra sobre estudo, permite aumentar o grau de eficiência e usabilidade da abordagem em questão. A identificação e resolução de conflitos são um meio de providenciar uma forma de satisfazer as necessidades e prioridades dos clientes no sistema a desenvolver, conferindo-lhes o poder de decisão. A existência de mecanismos que possibilitem a composição entre requisitos *crosscutting* e não *crosscutting* fornece a possibilidade de reagrupar todos os elementos anteriormente separados e recuperar as suas interligações.

4.2.1. Aplicar os critérios em abordagens EROA

Como a abordagem de Problem Frames (PF) integrada com AORE proposta nesta dissertação de mestrado incorpora conceitos aspectuais, esta reúne condições para ser

comparada, utilizando os critérios acima mencionados, com outras abordagens orientadas a aspectos.

A tabela 4.17 apresenta a realização dessa comparação com AORE [22], Cenários com aspectos [28], VAODA [23], MATA [29], AORA [4] e Vision [2].

Tabela 4.17: Problem Frames integrado com AORE vs abordagens EROA.

Critério \ Abordagem		PF e AORE	AORE	Cenários com aspectos	VAODA	MATA	AORA	Vision e use cases
Requisitos funcionais		VPs, PF	VPs	DS	VPs	DS	UC	VPs / UC
Requisitos não funcionais		Sim	Sim	Sim	Sim	Não	Sim	Sim
Crosscutting concerns	Funcionais	Sim	Não	Sim	Sim	Sim	Sim	Sim
	Não funcionais	Sim	Sim	Sim	Não	Não	Sim	Sim
Ferramenta de suporte		Não	ARCaDe	Não	Aspect Finder	AGG	Sim	Não
Resolução de conflitos		Sim	Sim	Não	Não	Não	Sim	Sim
Mecanismos de composição		Sim	Sim	Sim	Sim	Sim	Sim	Não

(VPs: Viewpoints**; PF:** Problem Frames**; DS:** Diagramas de sequência**; UC:** Use cases**)**

Todas as abordagens identificam e especificam os requisitos funcionais, seja através de *viewpoints*, diagramas de sequência ou *use cases*. A nossa abordagem utiliza, no entanto, *viewpoints* e Problem Frames. O mesmo não acontece com os requisitos não funcionais, pois MATA não lida com este tipo de requisitos.

À excepção de AORE, que não captura os requisitos funcionais *crosscutting*, todas as restantes abordagens o fazem. Já os *crosscutting concerns* não funcionais, apenas não são contemplados nas técnicas de VAODA e MATA.

AORE utiliza a ferramenta ARCaDe [22], VAODA recorre ao uso de Aspect Finder [23], enquanto MATA faz uso de AGG [29].

Apenas a abordagem integrada apresentada nesta tese possibilita a utilização de Problem Frames para representação de um problema de software.

Quanto à resolução de conflitos, e como foram integrados alguns conceitos provenientes de AORE, esta abordagem inclui também essa particularidade. Também as abordagens AORA e Vision resolvem as situações potencialmente conflituosas.

Resta apenas verificar que, exceptuando Vision, todas as abordagens acima mencionadas contêm mecanismos de composição de aspectos.

4.2.2. Aplicar os critérios em abordagens de Problem Frames integrado com aspectos

A comparação efectuada nesta secção é exclusiva de abordagens que integrem a capacidade de identificação e modularização de aspectos em Problem Frames (PF).

A tabela 4.18 apresenta o comparativo entre as várias abordagens, nomeadamente Problem Frames e aspectos [18], Problem Frames e cenários aspectuais [17] e requisitos de segurança em Problem Frames [10].

Tabela 4.18: Problem Frames integrado com AORE vs abordagens PF com aspectos**.**

Critério \ **Abordagem**		**PF e AORE**	**PF e aspectos**	**PF e cenários aspectuais**	**Segurança em PF**
Requisitos funcionais		VPs, PF	PF	PF, cenários	PF
Requisitos não funcionais		Sim	Não	Não	Sim
Crosscutting concerns	Funcionais	Sim	Sim	Sim	Não
	Não funcionais	Sim	Não	Não	Sim
Ferramenta de suporte		Não	Não	Não	Não
Resolução de conflitos		Sim	Não	Não	Sim
Mecanismos de composição		Sim	Sim	Sim	Sim

(VPs: Viewpoints**; PF:** Problem Frames**)**

Como todas as abordagens aqui utilizadas para a comparação possuem Problem Frames, o critério de requisitos funcionais diz respeito, essencialmente, à capacidade de organizar e estruturar este tipo de requisitos previamente à representação do problema em forma de diagramas.

A abordagem de Problem Frames e aspectos [18], como o nome indica, suporta a detecção e modularização de aspectos, embora, apenas os de natureza funcional. Apenas a abordagem de Problem Frames integrada com AORE apresentada nesta dissertação consegue reconhecer, tratar e representar *crosscutting concerns* não funcionais.

Nenhuma das abordagens aqui referidas possui uma ferramenta de suporte à modelação, no entanto, todas elas contêm Problem Frames.

Todas as abordagens contemplam mecanismos de composição de aspectos, através de regras específicas, embora apenas aplicadas ao tipo de aspectos que cada uma consegue identificar e separar. A resolução de conflitos existe apenas, para além da nossa abordagem, em Vision [2].

5. Conclusões

Problem Frames é uma técnica de engenharia de requisitos que permite analisar e descrever um problema de software. No entanto, há que ter em consideração, que alguns dos requisitos, os quais se pretendem estudar e especificar, podem ser considerados transversais (*crosscutting*). Este factor pode dificultar o desenvolvimento e manutenção de um software, pois cada um destes elementos pode afectar diversas partes do sistema.

É para ultrapassar este inconveniente, que a área de engenharia de requisitos orientada a aspectos (EROA) procura detectar logo nas fases iniciais do ciclo de vida do software, as propriedades transversais existentes.

Esta dissertação de Mestrado propôs uma abordagem que consegue identificar e tratar de forma eficaz os requisitos aspectuais, por intermédio duma técnica AORE baseada em *viewpoints* [22] que integra o modelo de Problem Frames [13] de modo a permitir, não só, a representação do problema em si, mas também de todos os aspectos (funcionais e não funcionais).

A abordagem final é composta por cinco partes fundamentais. A Identificação e especificação de requisitos que é composta por duas actividades: Identificação de *viewpoints* e especificação de requisitos e Identificação e especificação de *concerns*.

A Identificação de aspectos não funcionais é formada por três actividades: Relacionar *viewpoints* e *concerns*, Identificar *crosscutting concerns* e Identificação e resolução de conflitos.

A Identificação de aspectos funcionais é constituída por três actividades: Identificação de funcionalidades e requisitos, Relacionar *viewpoints*, requisitos e funcionalidades e Identificar aspectos funcionais.

A parte de Problem Frames consiste em cinco passos onde são especificados o Diagrama de Contexto, Diagramas de problema, Diagramas de problema aspectuais, Operacionalização de aspectos não funcionais e Decomposição e especificação de problemas.

Por último, as Regras de composição de aspectos divide-se em: Regras de composição de aspectos funcionais e Regras de composição de aspectos não funcionais.

A abordagem integrada foi ainda aplicada aos casos de estudo do parque de estacionamento e Health Watcher, este último mais exaustivamente.

5.1. Contribuições

A abordagem presente nesta dissertação engloba os conceitos de uma técnica direccionada à elicitação de requisitos, que contempla Problem Frames e aspectos.

Uma das principais contribuições da abordagem consiste em possibilitar a obtenção de diagramas de problemas já tendo em conta e modularizando as características aspectuais, quer estas sejam funcionais ou não funcionais.

Foram ainda elaboradas regras de composição de aspectos que visam a associação entre os elementos aspectuais e não aspectuais a nível de *problem frames*.

A integração realizada teve como base, a definição de um meta-modelo de AORE [22], adaptando-o para Problem Frames [13], por intermédio da detecção de aspectos funionais e representando também os vários tipos de *concerns*. Assim, possibilitou o seu relacionamento com o meta-modelo de Problem Frames de [11, 18], de maneira a garantir a integridade e coerência das relações desta abordagem. Foram definidas regras de restrição OCL ao modelo integrado, garantindo assim, a integridade das relações estabelecidas entre os seus elementos e um ponto de partida para a implementação de uma ferramenta de suporte a esta técnica.

5.2. Limitações

Uma das limitações desta abordagem é não possuir uma ferramenta que suporte a sua implementação.

Também a linguagem de composição de aspectos pode ser alvo de evolução, devido à sua simplicidade.

O meta-modelo de Problem Frames integrado com AORE não contempla ainda problemas compostos, não especificando totalmente os relacionamentos existentes com os outros elementos.

5.3. Trabalho futuro

Como trabalho futuro é importante colmatar algumas das limitações da abordagem, nomeadamente, providenciar uma ferramenta que possibilite o desenvolvimento e faça a correspondência entre os *viewpoints* e requisitos aspectuais (funcionais e não funcionais) com a sua representação em diagramas de problema.

Melhorar a linguagem de composição de modo a permitir uma melhor caracterização e uma forma mais completa de restabelecer as ligações entre os diagramas de problema aspectuais e os restantes.

Por último, completar o meta-modelo integrado para possibilitar uma forma mais correcta de utilizar problemas compostos na especificação de um problema, tendo em conta a presença de aspectos.

Bibliografia

1. **Araújo J. e Coutinho P.** Identifying Aspectual Use Cases Using a Viewpoint-Oriented Requirements Method. Early Aspects 2003: Aspect-Oriented Requirements Engineering and Architecture Design, Workshop of the 2nd International Conference on Aspect-Oriented Software Development, Boston, USA, 2003.

2. **Araújo J. e Coutinho P.** Identifying Aspectual Use Cases Using a Viewpoint-Oriented Requirements Method.

3. **Araújo J., Whittle J. e Kim D.** Modeling and composing scenario-based requirements with aspects. Presented at 12th IEEE Int. Requirements Engineering Conf. (RE), Kyoto, Japão, 2004.

4. **Brito I., Vieira F., Moreira A. e Ribeiro R.** Handling conflicts in aspectual requirements compositions. Transactions on Aspect-Oriented software development, 2007, Vol. III, LNCS 4620.

5. **Brito I. e Moreira A.** Towards a Composition Process for Aspect-Oriented Requirements. Early Aspects 2003: Aspect-Oriented Requirements Engineering and Architecture Design, Boston, EUA, 2003.

6. **Chitchyan R., Rashid A., Sawyer P., Garcia A., Alarcon M. P., Bakker J., Tekinerdogan B., Clarke S. e Jackson A.** Survey of Analysis and Design Approaches, 2005.

7. **Chung L., Nixon B., Yu E. e Mylopoulos J.** Non-functional Requirements in Software Engineering. Kluwer Academic Publishers, 2000.

8. **Cox K., Hall Jon G. e Rapanotti L.** A roadmap of problem frames research. Information and Software Technology 47, 2005, pp. 891–902.

9. **Filman E., Elrad T., Clarke S. e Aksit M.** Aspect-Oriented Software Development, Addison Wesley, 2004.

10. **Haley B., Laney C. e Nuseibeh B.** Deriving Security Requirements from Crosscutting Threat Descriptions. Proceedings of the 3rd International Conference on Aspect-Oriented Software Development, Lancaster, UK, 2004, pp. 112-121.

11. **Hatebur D., Heisel M. e Schmidt H.** A Formal Metamodel for Problem Frames. 11th International Conference on Model Driven Engineering Languages and Systems, Toulouse, France, 2008.

12. **Jackson M.** Problem Frames and Software Engineering. Information and Software Technology 47, 2005, pp. 903–912.

13. **Jackson M.** Problem Frames: Analyzing and Structuring Software Development Problems. Addison Wesley, 2001.

14. **Jacobson I. e Pan-Wei Ng** Aspect-Oriented Software Development with Use Cases. Addison Wesley, 2004.

15. **Kotonya G. e Sommerville I.** Requirements Engineering With Viewpoints. BCS/IEE Software Engineering Journal, 1996, pp. 5-18.

16. **Laney R., Barroca L., Jackson M. e Nuseibeh B.** Composing Requirements Using Problem Frames. Requirements Engineering Conference, Kyoto, Japan, 2004.

17. **Lencastre M., Araújo J., Moreira A. e Castro J.** Aspects Composition in Problem Frames, 2008.

18. **Lencastre M., Araújo J., Moreira A. e Castro J.** Towards Aspectual Problem Frames: an Example. Expert Systems Journal, Blackwell Publishing, 2008, Vol. 25, pp. 63-75.

19. **Moreira A., Araújo J. e Rashid A.** A Concern-Oriented Requirements Engineering Model. The 17th Conference on Advanced Information Systems Engineering (CAiSE'05), 2005, pp. 293-308, LNCS 3520.

20. OCL 2.0 Specification version 2.0, 2005, http://st.inf.tu-dresden.de/oclportal/.

21. Papyrus UML, 2008, http://www.papyrusuml.org.

22. **Rashid A., Moreira A. e Araújo J.** Modularisation and Composition of Aspectual Requirements. The 2nd International Conference on Aspect-Oriented Software Development(AOSD), 2003, pp. 11-20.

23. **Rodrigues A. P. e Araújo J.** VAODA: A viewpoint and aspect-oriented domain analysis Approach. International Conference on Enterprise Information Systems (ICEIS 2009), Milão, Itália, 2009.

24. **Rosenhainer L.** Identifying Crosscutting Concerns in Requirements Specification . Proceedings of OOPSLA Early Aspects 2004.

25. **Soares S., Laureano E. e Borba P.** Distribution and Persistence as Aspects. Software: Practice & Experience, 36(6), 2006.

26. **Sommerville I. e Sawyer P.** Requirements Engineering – A Good Practice Guide. John Wiley and Sons, 1997.

27. **Sommerville I.** Software Engineering 8. Addison Wesley, 2007.

28. **Whittle J. e Araújo J.** Scenario Modelling with aspects. IEE Proceedings - Software Special Issue, 2004, Vol. 151, pp. 157-172.

29. **Whittle J. e Jayaraman P.** MATA: A Tool for Aspect-Oriented Modeling based on Graph Transformation. Workshop on Aspect Oriented Modeling at MODELS 2007, Nashville, EUA, 2007.

30. XML - Extensible Markup Language, 2008, http://www.w3.org/XML/.

Apêndice A: problem frames elementares [13]

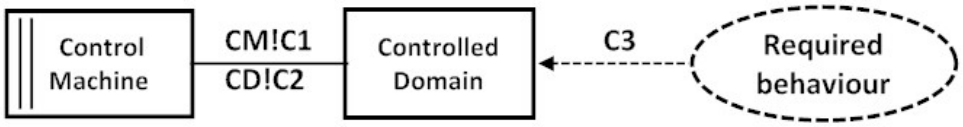


Figura A.1: Required Behaviour problem frame.

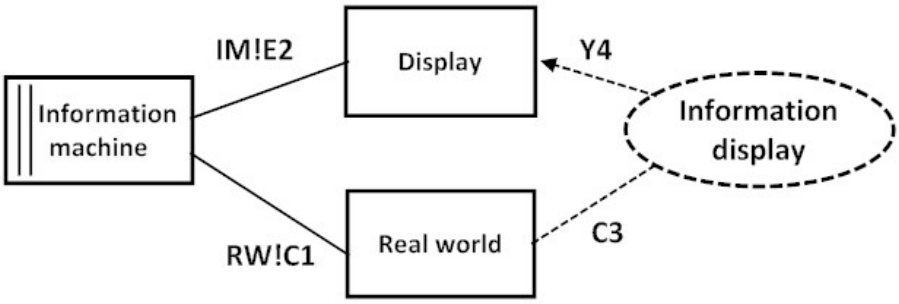


Figura A.2: Information display problem frame.

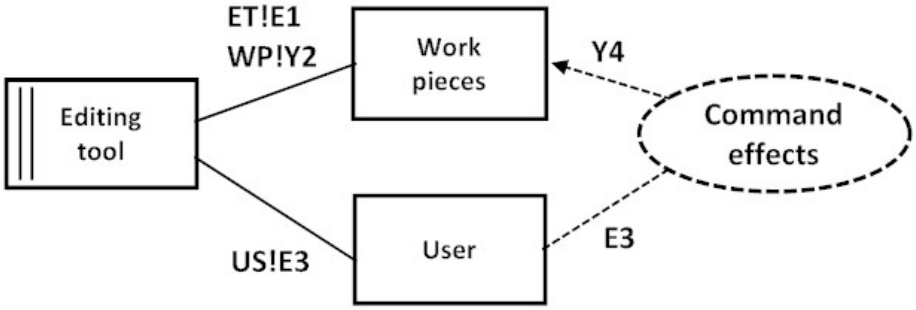


Figura A.3: Simple workpieces problem frame.

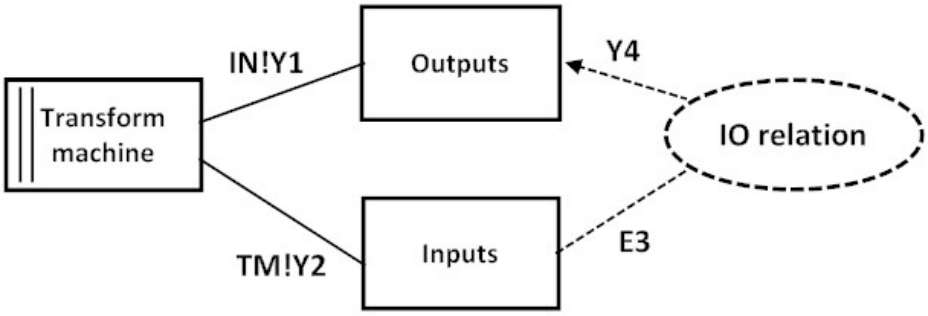


Figura A.4: Transformation problem frame.

Apêndice B: Meta-modelo de Problem Frames

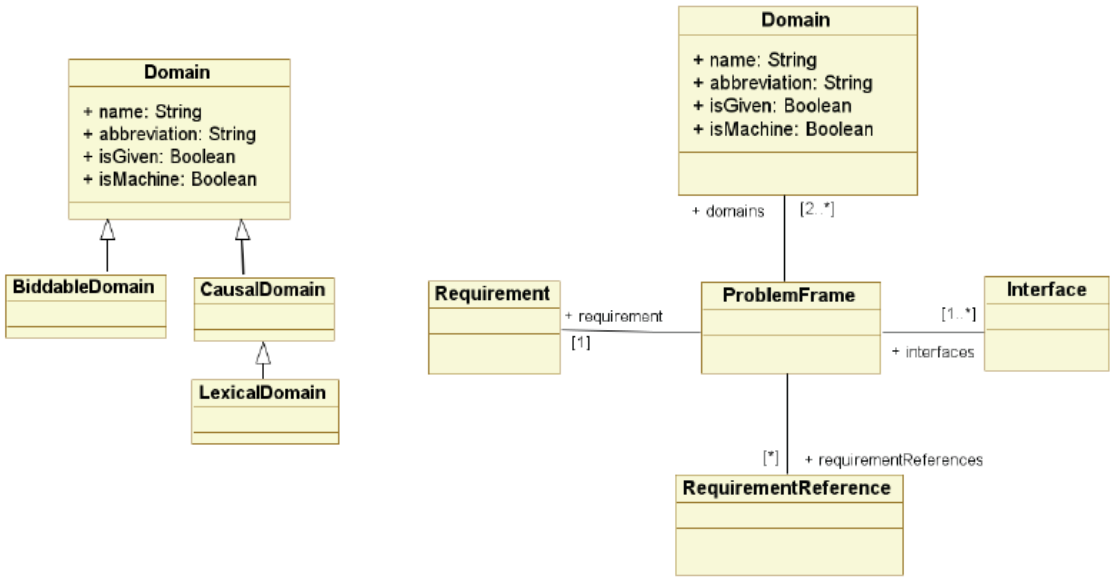
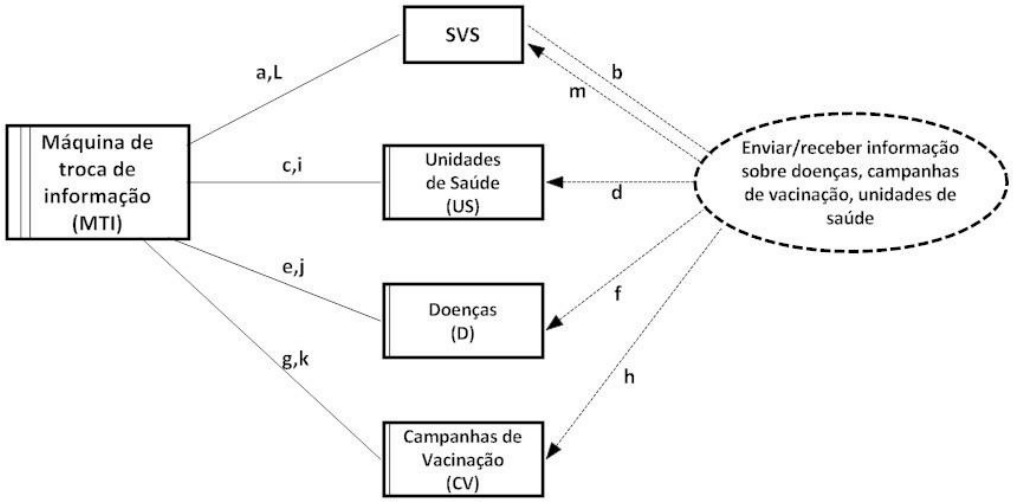


Figura B.1: Diagrama de classes UML do modelo de Problem Frames [11].

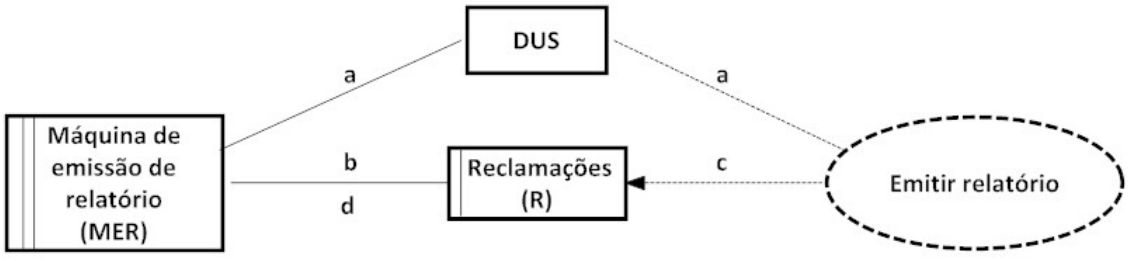
Apêndice C: Health-Watcher

Diagramas de problema



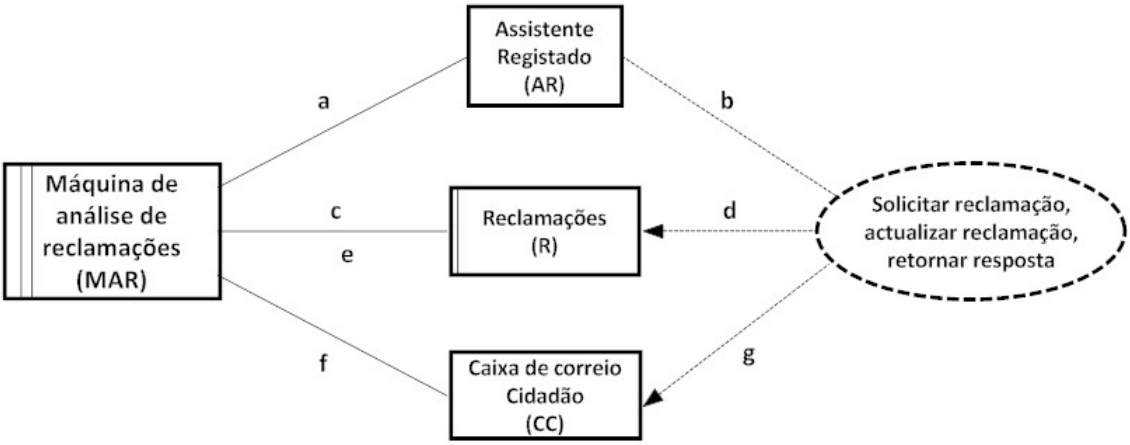
a: SVS!{informação(tipo), informação enviada com sucesso}
b: SVS!{informação, tipo da informação}
c: MTI!{actualizarUS(informação)}
d: US!{informação actualizada}
e: MTI!{actualizarDoenças(informação)}
f: D!{informação actualizada}
g: MTI!{actualizarCV(informação)}
h: CV!{informação actualizada}
i: US!{informação}
j: D!{informação}
k: CV!{informação}
L: MTI!{enviarInformação(informação)}
m: SVS!{informação enviada}

Figura C.1: Diagrama de problema Trocar informação.



a: DUS!{emitirRelatório()}
b: MER!{emitirRelatório(relatório)}
c: R!{relatório emitido}
d: R!{relatório}

Figura C.2: Diagrama de problema Emitir relatório.



a: AR!{obterReclamação(), actualizarReclamação(reclamação), enviarResposta(reclamação)}

b: AR!{reclamação, dados da análise, resposta da reclamação, obterReclamação()}

c: MAR!{obterReclamação(), actualizarReclamação(reclamação)}

d: R!{reclamação obtida, reclamação actualizada}

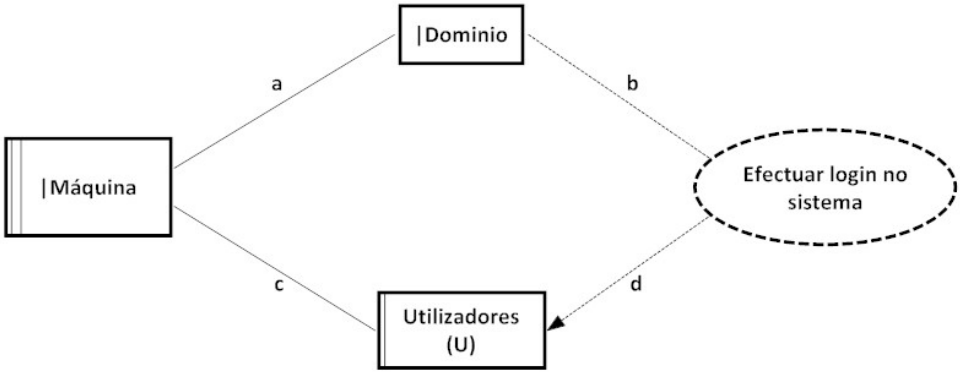
e: R!{reclamação}

f: MAR!{enviarResposta(reclamação, cidadão)}

g: CC!{resposta enviada}

Figura C.3: Diagrama de problema Analisar informação.

Diagramas de problema aspectuais com frame concern



a: Dominio!{login()}

b: Dominio!{utilizador, palavra chave}

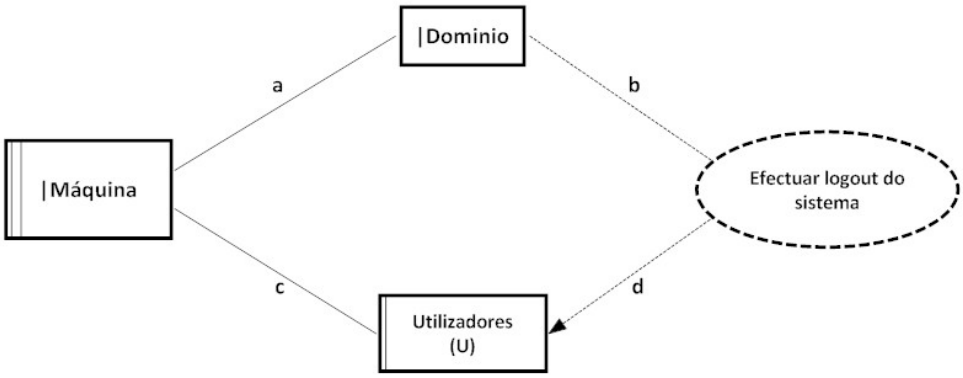
c: Máquina!{login(utilizador, palavra chave)}

d: U!{Login efectuado}

Figura C.4: Diagrama de problema aspectual Login (AspLogin).

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o |Dominio insere o utilizador e palavra chave e efectua o pedido de login() para a |Máquina.	Requisito
2. A |Máquina processa o comando login(utilizador, palavra chave).	Especificação da máquina
3. Resultando na alteração de estado do utilizador.	Descrição de domínio
4. Deste modo é alcançado o requisito Efectuar login com login efectuado.	Requisito

Figura C.5: Frame concern de AspLogin.



a: Dominio!{logout()}
b: Dominio!{utilizador}
c: Máquina!{logout(utilizador)}
d: U!{Logout efectuado}

Figura C.6: Diagrama de problema aspectual Logout (AspLogout).

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o |Dominio efectua o pedido de logout() para a |Máquina.	Requisito
2. A |Máquina processa o comando logout(utilizador).	Especificação da máquina
3. Resultando na alteração de estado do utilizador.	Descrição de domínio
4. Deste modo é alcançado o requisito Efectuar logout com logout efectuado.	Requisito

Figura C.7: Frame concern de AspLogout.

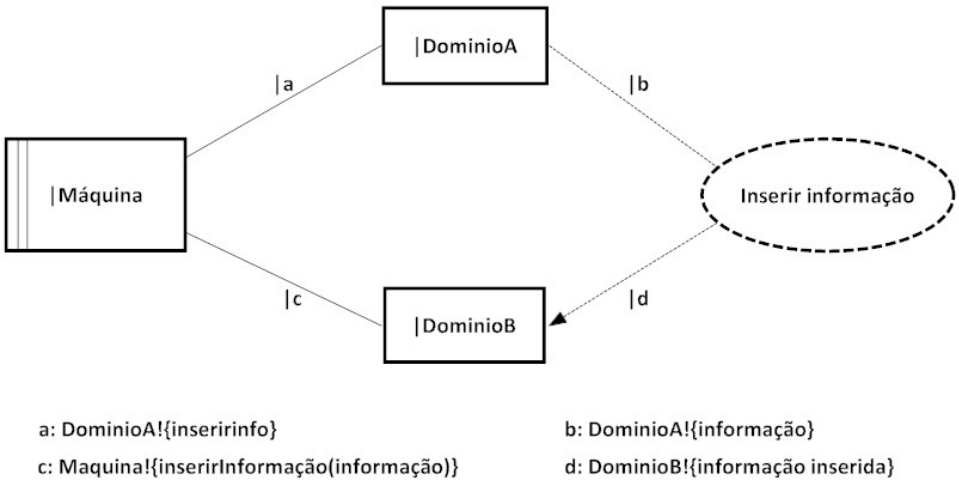


Figura C.8: Diagrama de problema aspectual Inserir informação (AspInserirInfo).

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o |DominioA efectua inseririnfo para a |Máquina.	Requisito
2. A |Máquina processa o comando inserirInformação(informação).	Especificação da máquina
3. Resultando na inserção da informação no |DominioB.	Descrição de domínio
4. Deste modo é alcançado o requisito Inserir informação com informação inserida.	Requisito

Figura C.9: Frame concern de AspInserirInfo.

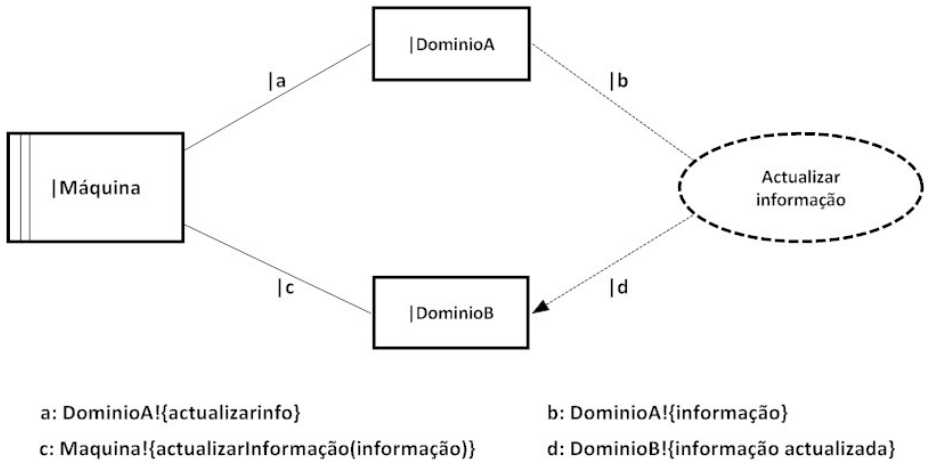


Figura C.10: Diagrama de problema aspectual Actualizar informação (AspActualizarInfo).

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o |DominioA efectua actualizarinfo para a |Máquina.	Requisito
2. A |Máquina processa o comando actualizarInformação(informação).	Especificação da máquina
3. Resultando na actualização da informação no |DominioB.	Descrição de domínio
4. Deste modo é alcançado o requisito Actualizar informação com informação actualizada.	Requisito

Figura C.11: Frame concern de AspActualizarInfo.

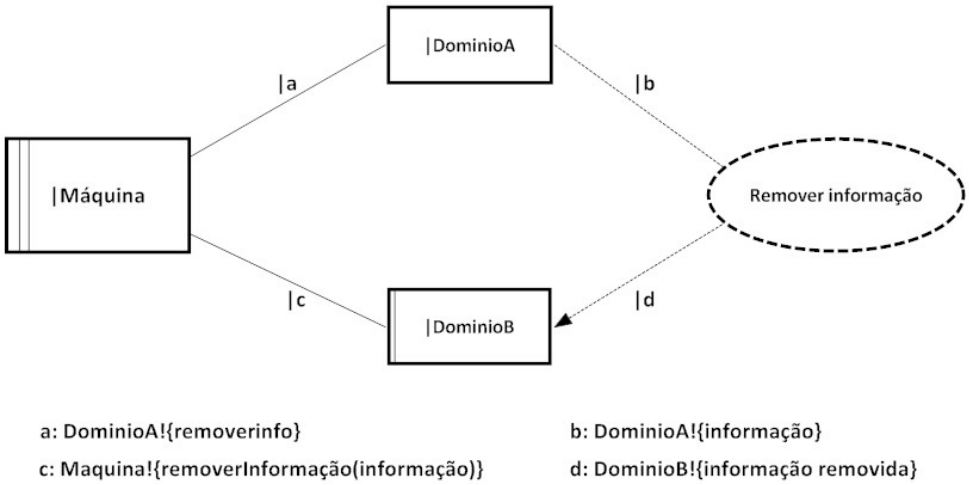
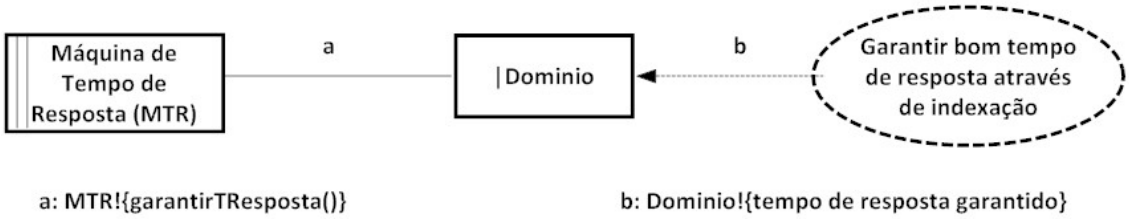


Figura C.12: Diagrama de problema aspectual Remover informação (AspRemoverInfo).

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o |DominioA efectua removerinfo para a |Máquina.	Requisito
2. A |Máquina processa o comando removerInformação(informação).	Especificação da máquina
3. Resultando na remoção da informação do |DominioB.	Descrição de domínio
4. Deste modo é alcançado o requisito Remover informação com informação removida.	Requisito

Figura C.13: Frame concern de AspRemoverInfo.

Diagramas de problema aspectuais não funcionais com frame concern

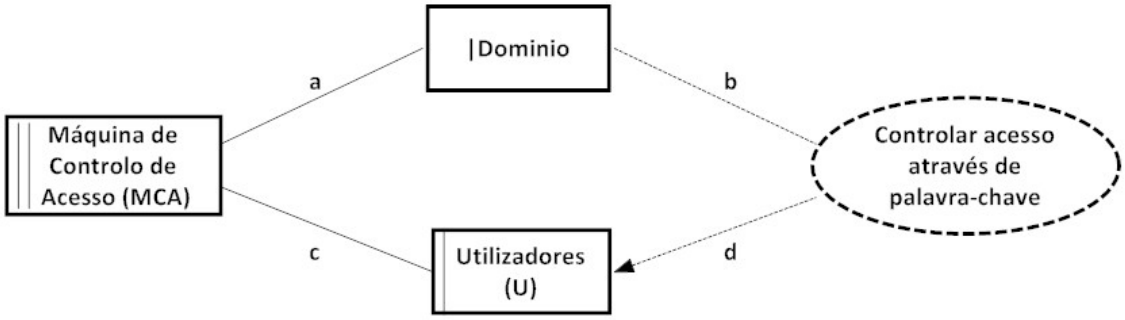


a: MTR!{garantirTResposta()}

b: Dominio!{tempo de resposta garantido}

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. A MTR processa o comando garantirTResposta().	Especificação da máquina
2. Resultando na numa melhor eficiência em |Dominio, através da sua indexação.	Descrição de domínio
3. Deste modo é alcançado o requisito Garantir tempo de resposta com novo tempo de resposta garantido.	Requisito

Figura C.14: Diagrama de problema aspectual não funcional de Tempo de Resposta e frame concern.



a: Dominio!{login()}

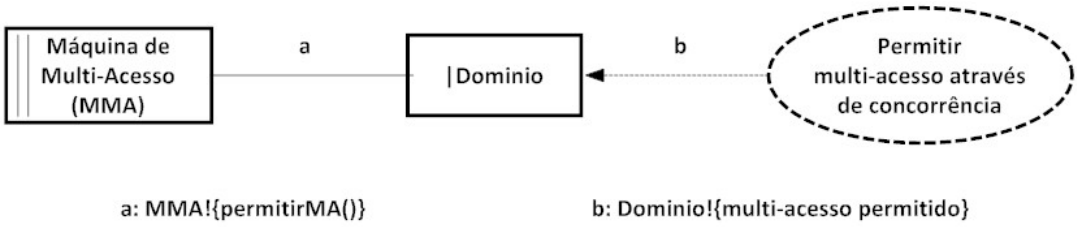
b: Dominio!{utilizador, palavra chave}

c: MCA!{verificarPalavra(utilizador, palavra-chave)}

d: U!{acesso permitido}

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o |Dominio introduz o utilizador e palavra chave e efectua login() para a MCA.	Requisito
2. A MCA processa o comando verificarPalavra(utilizador, palavra chave).	Especificação da máquina
3. Resultando na permissão de acesso em Utilizadores.	Descrição de domínio
4. Deste modo é alcançado o requisito Controlar acesso através de acesso permitido.	Requisito

Figura C.15: Diagrama de problema aspectual não funcional de Segurança e frame concern.



Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. A MMA processa o comando permitirMA().	Especificação da máquina
2. Resultando na alteração de |Dominio passando a permitir multi-acesso com concorrência.	Descrição de domínio
3. Deste modo é alcançado o requisito Permitir multi-acesso com multi-acesso permitido.	Requisito

Figura C.16: Diagrama de problema aspectual não funcional de Multi-acesso e frame concern.

Diagramas de problema decompostos com frame concern

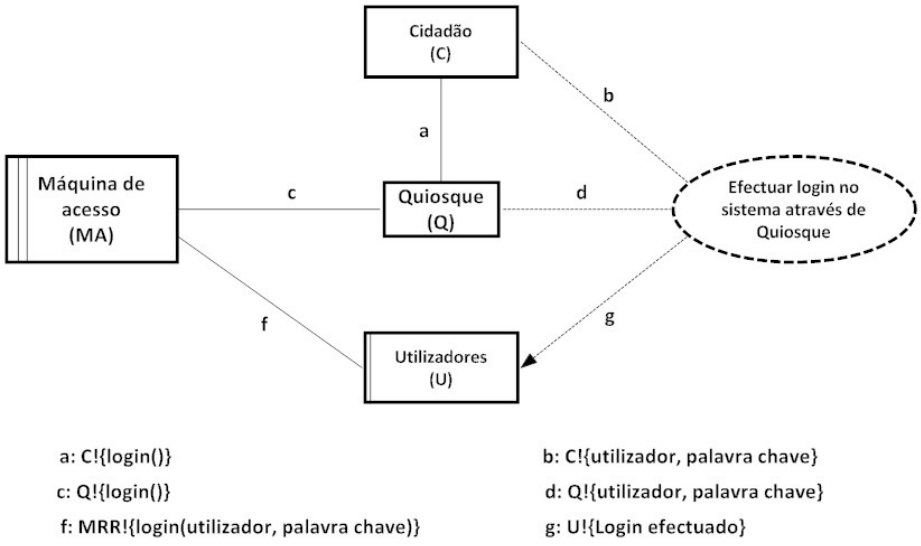


Figura C.17: Diagrama de problema Login através de Quiosque.

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o Cidadão introduz o utlizador e palavra-chave e efectua login() ao Quiosque, este faz o pedido de login() à MA.	Requisito
2. A MA acede à base de dados Utilizadores e efectua o comando login(utilizador, palavra chave).	Especificação da máquina
3. Resultando na mudança do estado do utilizador para registado no sistema.	Descrição de domínio
4. Deste modo é alcançado o requisito Login através de Quiosque com Login efectuado.	Requisito

Figura C.18: Frame concern de Login através de Quiosque.

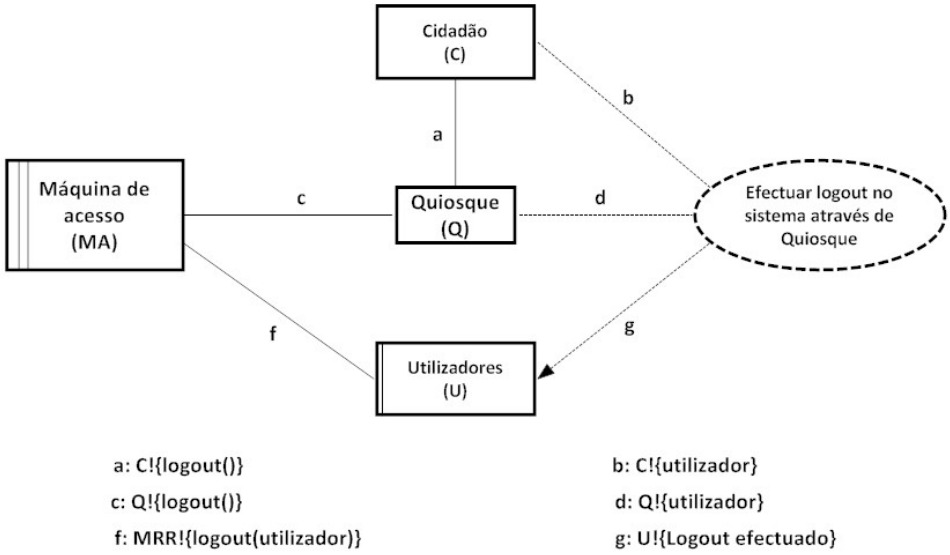


Figura C.19: Diagrama de problema Logout através de Quiosque.

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o Cidadão introduz a reclamação e efectua registarReclamação().	Requisito
2. A MRR acede à base de dados Reclamções e efectua o comando registarReclamação(reclamação).	Especificação da máquina
3. Resultando na ciração de uma nova reclamação.	Descrição de domínio
4. Deste modo é alcançado o requisito Registar reclamação com reclamação registada.	Requisito

Figura C.20: Frame concern de Logout através de Quiosque.

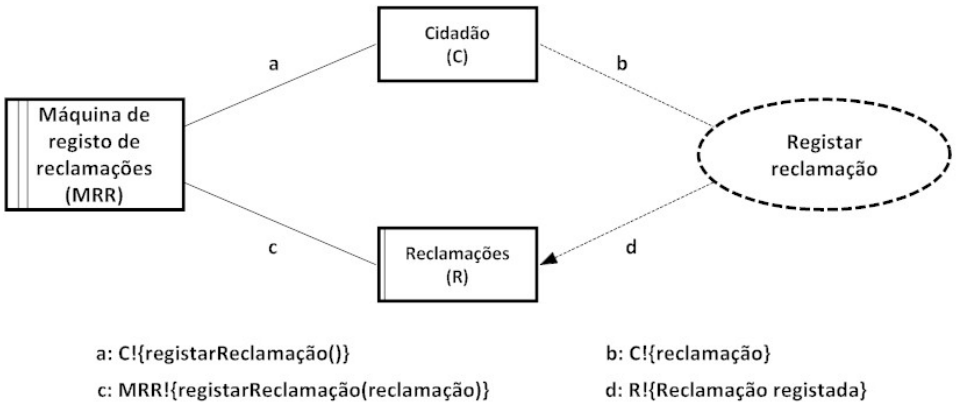


Figura C.21: Diagrama de problema Registar reclamação.

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o Cidadão introduz a reclamação e efectua registarReclamação() no Quiosque, este faz o pedido de registarReclamação() à MRR.	Requisito
2. A MRR acede à base de dados Reclamções e efectua o comando registarReclamação(reclamação).	Especificação da máquina
3. Resultando na ciração de uma nova reclamação.	Descrição de domínio
4. Deste modo é alcançado o requisito Registar reclamação através de Quiosque com reclamação registada.	Requisito

Figura C.22: Frame concern de Registar reclamação.

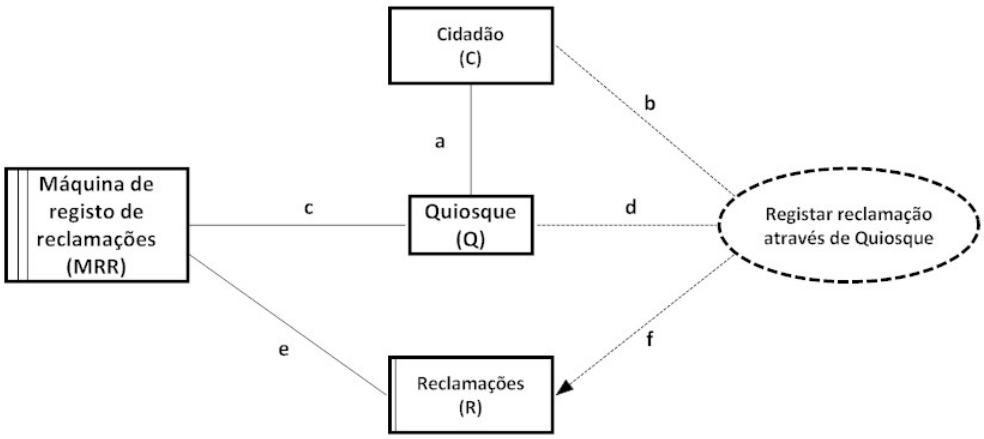
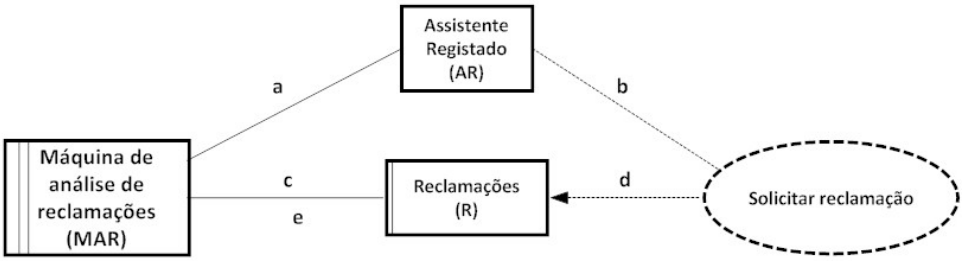


Figura C.23: Diagrama de problema Registar reclamação através de Quiosque.

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o Cidadão introduz a reclamação e efectua registarReclamação() no Quiosque, este faz o pedido de registarReclamação() à MA.	Requisito
2. A MA acede à base de dados Reclamções e efectua o comando registarReclamação(reclamação).	Especificação da máquina
3. Resultando na ciração de uma nova reclamação.	Descrição de domínio
4. Deste modo é alcançado o requisito Registar reclamação através de Quiosque com reclamação registada.	Requisito

Figura C.24: Frame concern de Registar reclamação através de Quiosque.

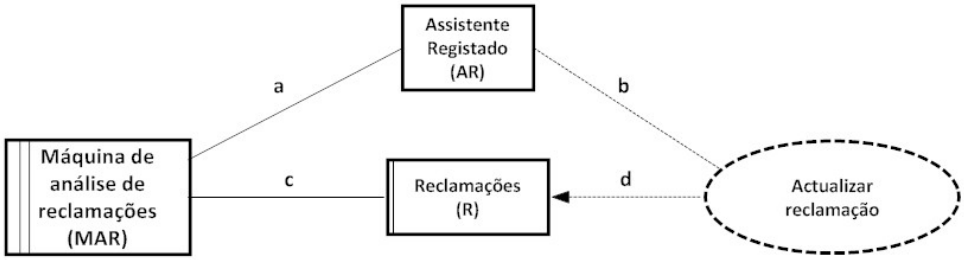


a: AR!{obterReclamação()} b: AR!{obterReclamação()}
c: MAR!{obterReclamação()} d: R!{reclamação obtida}
e: R!{reclamação}

Figura C.25: Diagrama de problema Solicitar informação.

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o AR efectua obterReclamação() à MAR.	Requisito
2. A MAR acede à base de dados Reclamações e efectua o comando obterReclamação().	Especificação da máquina
3. Resultando na obtenção da reclamação e mudança do seu estado para aberta.	Descrição de domínio
4. Deste modo é alcançado o requisito Solicitar reclamação com reclamação obtida.	Requisito

Figura C.26: Frame concern de Solicitar informação.



a: AR!{actualizarReclamação(reclamação)} b: AR!{reclamação, dados da análise}
c: MAR!{actualizarReclamação(reclamação)} d: R!{reclamação actualizada}

Figura C.27: Diagrama de problema Actualizar reclamação.

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o AR insere os dados da análise e efectua actualizarReclamação(reclamação) à MAR.	Requisito
2. A MAR acede à base de dados Reclamações e efectua o actualizarReclamação(reclamação).	Especificação da máquina
3. Resultando na alteração do estado da reclamação para analisada.	Descrição de domínio
4. Deste modo é alcançado o requisito Actualizar reclamação com reclamação actualizada.	Requisito

Figura C.28: Frame concern de Actualizar reclamação.

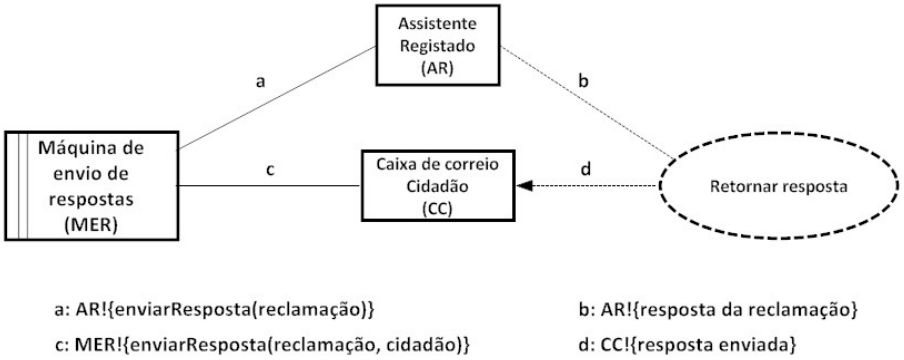
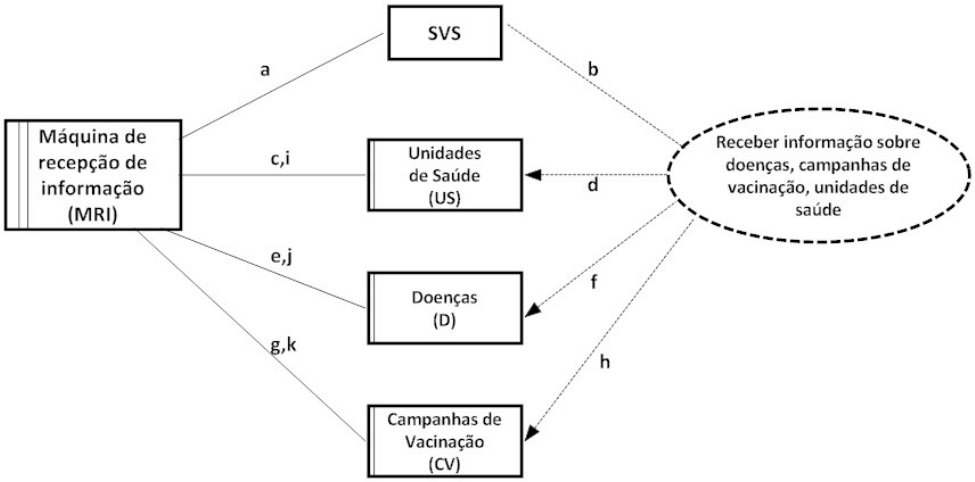


Figura C.29: Diagrama de problema Retornar resposta.

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o AR insere a resposta da reclamação e efectua enviarResposta(reclamação) à MER.	Requisito
2. A MER procede ao envio da resposta para CC através de enviarResposta(reclamação, cidadão).	Especificação da máquina
3. Resultando no envio da resposta para CC.	Descrição de domínio
4. Deste modo é alcançado o requisito Retornar resposta através de resposta enviada.	Requisito

Figura C.30: Frame concern de Retornar resposta.

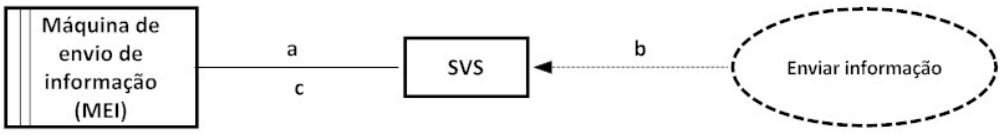


a: SVS!{informação(tipo)}
b: SVS!{informação, tipo da informação}
c: MRI!{actualizarUS(informação)}
d: US!{informação actualizada}
e: MRI!{actualizarDoenças(informação)}
f: D!{informação actualizada}
g: MRI!{actualizarCV(informação)}
h: CV!{informação actualizada}
i: US!{informação}
j: D!{informação}
k: CV!{informação}

Figura C.31: Diagrama de problema Receber informação.

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o SVS envia informação(tipo) à MRI.	Requisito
2. A MRI, dependendo do tipo de informação, acede às bases de dados US, D ou CV e efectua o comando actualizarUS(informação), actualizarDoenças(informação) ou actualizarCV(informação).	Especificação da máquina
3. Resultando na alteração da informação.	Descrição de domínio
4. Deste modo é alcançado o requisito Receber informação através de informação actualizada.	Requisito

Figura C.32: Frame concern de Receber informação.

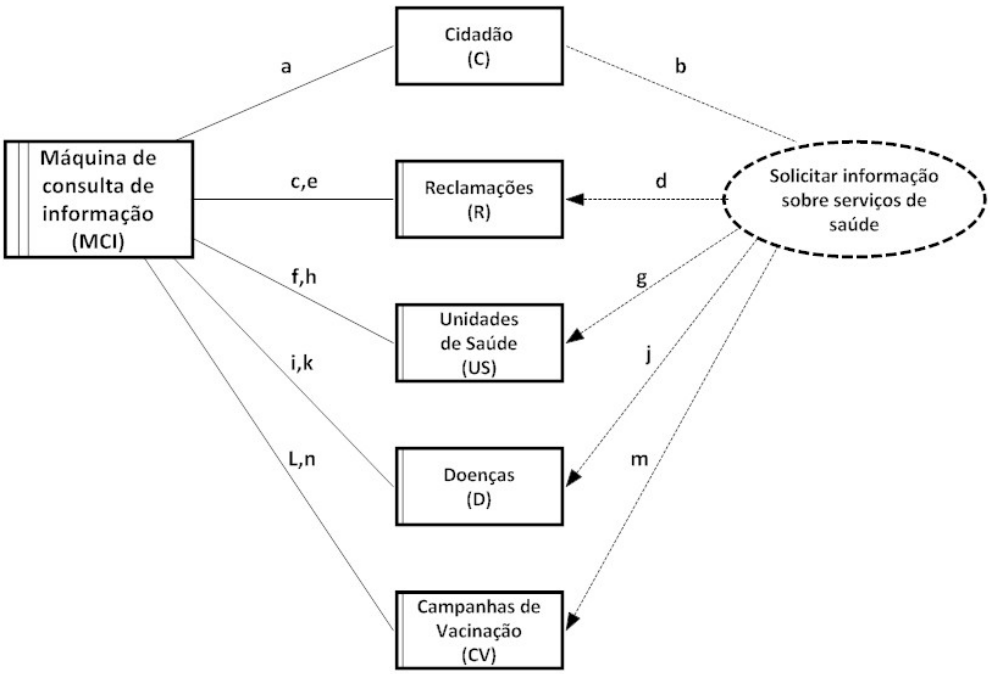


a: MEI!{enviarInformação(informação)}
b: SVS!{informação enviada}
c: SVS!{informação enviada com sucesso}

Figura C.33: Diagrama de problema Enviar informação.

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1.A MEI envia a informação para o SVS através de enviarInformação(informação).	Especificação da máquina
2. Resultando na recepção e confirmação que a informação foi enviada com sucesso.	Descrição de domínio
3. Deste modo é alcançado o requisito Enviar informação com informação enviada.	Requisito

Figura C.34: Frame concern de Enviar informação.



a: C!{informação(tipo)}
b: C!{tipo da informação}
c: MCI!{consultarReclamação(reclamação)}
d: R!{reclamação existente}
e: R!{reclamação}
f: MCI!{consultarUS(especialidade), consultarUS(unidadeSaúde)}
g: US!{unidade saúde existente, especialidade existente}
h: US!{unidade saúde, especialidade}
i: MCI!{consultarDoença(doença)}
j: D!{doença existente}
k: D!{doença}
L: MCI!{consultarVacinação(campanha)}
m: CV!{campanha vacinação existente}
n: CV!{campanha vacinação}

Figura C.35: Diagrama de problema Solicitar informação.

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o Cidadão efectua o pedido informação(tipo) à MCI.	Requisito
2. A MCI, dependendo do tipo de informação acede às bases de dados Reclamações,US, D ou CV e consulta a informação.	Especificação da máquina
3. Resultando na confirmação e obtenção da informação.	Descrição de domínio
4. Deste modo é alcançado o requisito Solicitar informação através de informação existente.	Requisito

Figura C.36: Frame concern de Solicitar informação.

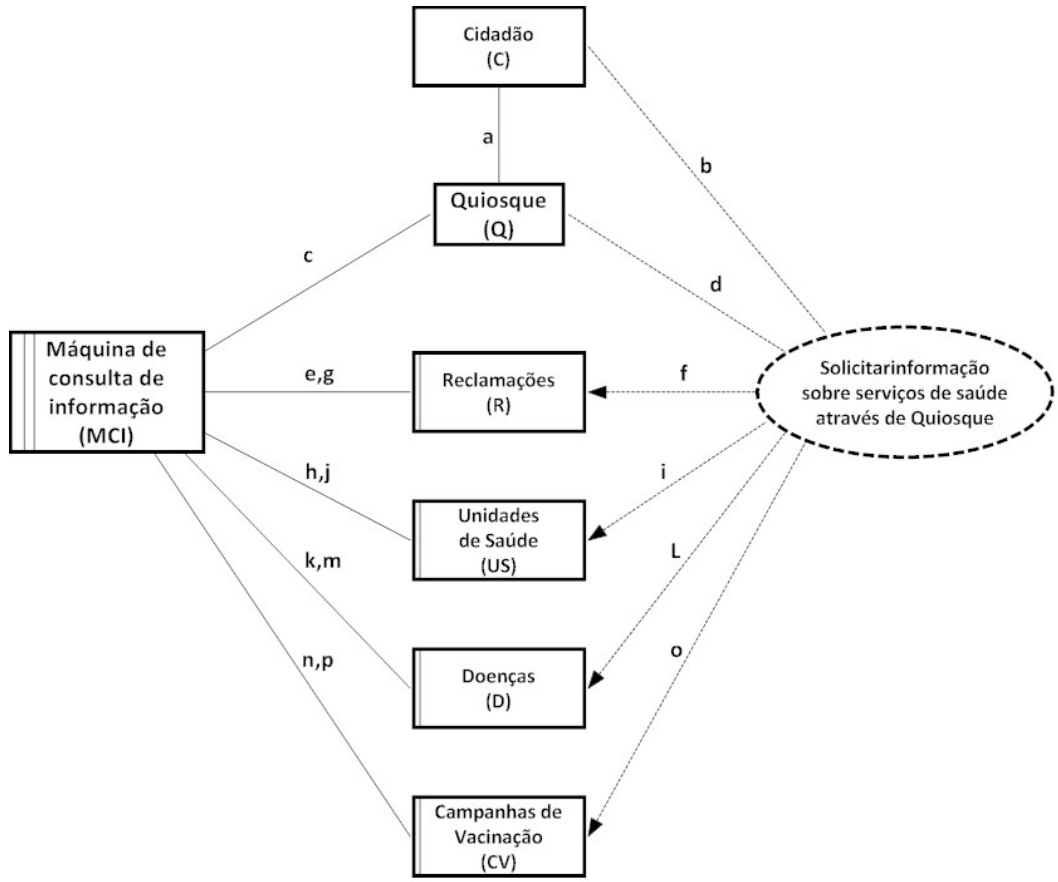
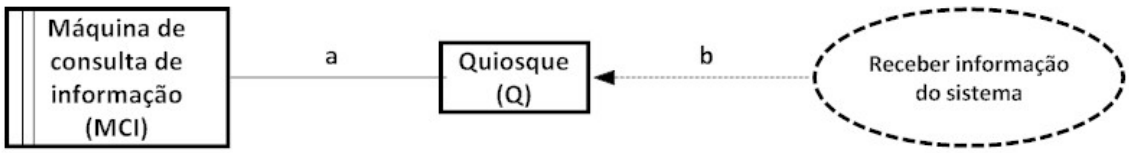


Figura C.37: Diagrama de problema Solicitar informação através de Quiosque.

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. Quando o Cidadão solicita informação ao Quiosque, que por sua vez efectua o pedido informação(tipo) à MCI.	Requisito
2. A MCI, dependendo do tipo de informação acede às bases de dados Reclamações,US, D ou CV e consulta a informação.	Especificação da máquina
3. Resultando na confirmação e obtenção da informação.	Descrição de domínio
4. Deste modo é alcançado o requisito Solicitar informação através de Quiosque com informação existente.	Requisito

Figura C.38: Frame concern de Solicitar informação através de Quiosque.



a: MCI!{receberInformação(informação)} b: Q!{informação recebida}

Argumentação do *Frame Concern* *Descrição Ordenada*	Tipo de descrição
1. A MCI envia a informação para o Quiosque através de receberInformação(informação).	Especificação da máquina
2. Resultando na recepção da informação.	Descrição de domínio
3. Deste modo é alcançado o requisito Receber informação através de informação recebida.	Requisito

Figura C.39: Diagrama de problema Receber informação do sistema e frame concern.

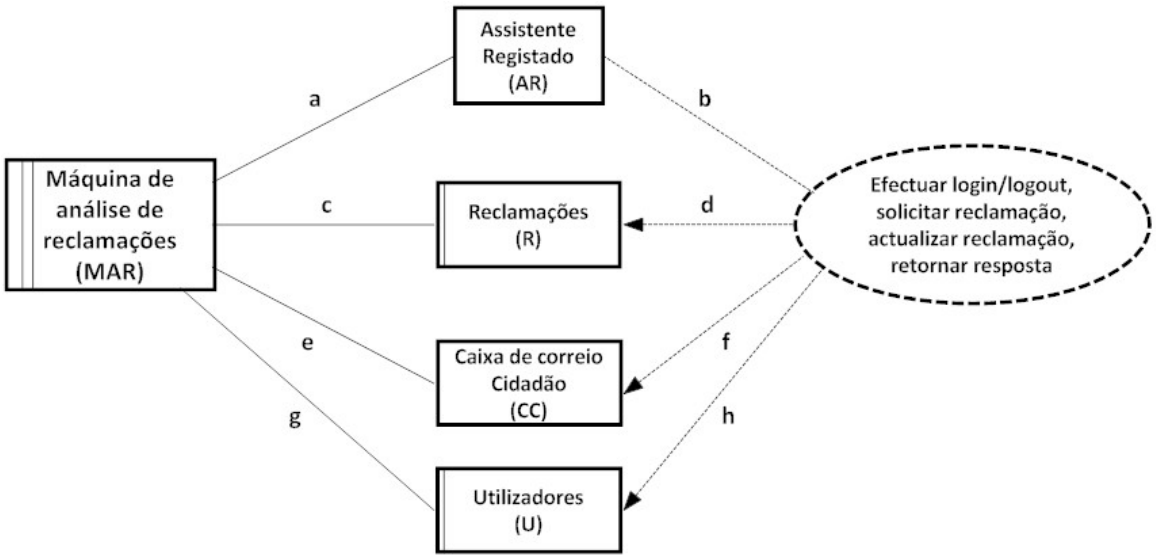
Diagramas de problema compostos

```
Compose aspect AspLogin with Analisar reclamação
Bind domain |Máquina to Máquina de análise de reclamações
Bind domain |Dominio to Assistente Registado
```

Figura C.40: Composição de AspLogin a Analisar reclamação.

```
Compose aspect AspLogout with Analisar reclamação
Bind domain |Máquina to Máquina de análise de reclamações
Bind domain |Dominio to Assistente Registado
```

Figura C.41: Composição de AspLogout a Analisar reclamação.



a: AR!{login(), logout(), obterReclamação(), actualizarReclamação(reclamação), enviarResposta(reclamação)}

b: AR!{utilizador, palavra chave, reclamação, dados da análise, resposta da reclamação, obterReclamação()}

c: MAR!{obterReclamação(), actualizarReclamação(reclamação)}

d: R!{reclamação obtida, reclamação actualizada}

e: MAR!{enviarResposta(reclamação, cidadão)}

f: CC!{reclamação enviada}

g: MAR!{login(utilizador, palavra chave), logout(utilizador)}

Figura C.42: Diagrama de problema Analisar reclamação composto.

```
Compose aspect AspLogin with Administrar informação
Bind domain |Máquina to Máquina de administração de informação
Bind domain |Dominio to Administrador Unidades Saúde
```

Figura C.43: Composição de AspLogin a Administrar informação.

```
Compose aspect AspLogout with Administrar informação
Bind domain |Máquina to Máquina de administração de informação
Bind domain |Dominio to Administrador Unidades Saúde
```

Figura C.44: Composição de AspLogout a Administrar informação.

```
Compose aspect AspInserirInfo with Administrar informação
Bind domain |Máquina to Máquina de administração de informação
Bind domain |DominioA to Administrador Unidades Saúde
Bind domain |DominioB to Unidades de Saúde
Bind phenomenon |a to inserirInfo(informação)
Bind phenomenon |b to especialidade
Bind phenomenon |c to inserirEspecialidade(especialidade, unidade de saude)
Bind phenomenon |d to especialidade inserida

Compose aspect AspInserirInfo with Administrar informação
Bind domain |Máquina to Máquina de administração de informação
Bind domain |DominioA to Administrador Unidades Saúde
Bind domain |DominioB to Doenças
Bind phenomenon |a to inserirInfo(informação)
Bind phenomenon |b to doença
Bind phenomenon |c to inserirDoença(doença)
Bind phenomenon |d to doença inserida

Compose aspect AspInserirInfo with Administrar informação
Bind domain |Máquina to Máquina de administração de informação
Bind domain |DominioA to Administrador Unidades Saúde
Bind domain |DominioB to Campanhas de Vacinação
Bind phenomenon |a to inserirInfo(informação)
Bind phenomenon |b to campanha
Bind phenomenon |c to inserirCV(campanha)
Bind phenomenon |d to campanha inserida
```

Figura C.45: Composição de AspInserirInfo a Administrar informação.

```
Compose aspect AspActualizarInfo with Administrar informação
Bind domain |Máquina to Máquina de administração de informação
Bind domain |DominioA to Administrador Unidades Saúde
Bind domain |DominioB to Unidades de Saúde
Bind phenomenon |a to actualizarInfo(informação)
Bind phenomenon |b to especialidade
Bind phenomenon |c to actualizarEspecialidade(especialidade, unidade de saude)
Bind phenomenon |d to especialidade actualizada

Compose aspect AspActualizarInfo with Administrar informação
Bind domain |Máquina to Máquina de administração de informação
Bind domain |DominioA to Administrador Unidades Saúde
Bind domain |DominioB to Doenças
Bind phenomenon |a to actualizarInfo(informação)
Bind phenomenon |b to doença
Bind phenomenon |c to actualizarDoença(doença)
Bind phenomenon |d to doença actualizada
```

```
Compose aspect AspActualizarInfo with Administrar informação
Bind domain |Máquina to Máquina de administração de informação
Bind domain |DominioA to Administrador Unidades Saúde
Bind domain |DominioB to Campanhas de Vacinação
Bind phenomenon |a to actualizarInfo(informação)
Bind phenomenon |b to campanha
Bind phenomenon |c to actualizarCV(campanha)
Bind phenomenon |d to campanha actualizada
```

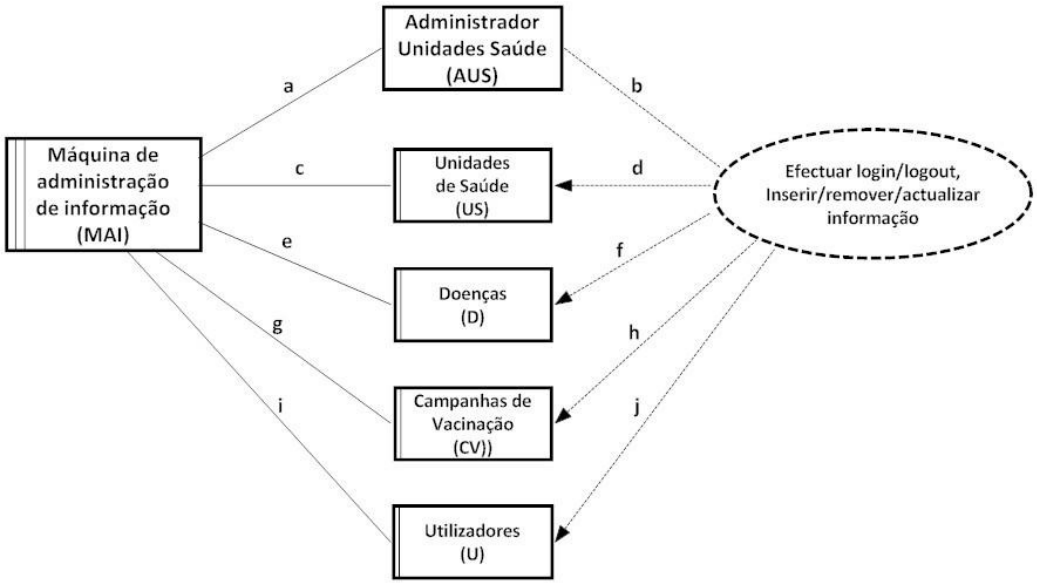
Figura C.46: Composição de AspActualizarInfo a Administrar informação.

```
Compose aspect AspRemoverInfo with Administrar informação
Bind domain |Máquina to Máquina de administração de informação
Bind domain |DominioA to Administrador Unidades Saúde
Bind domain |DominioB to Unidades de Saúde
Bind phenomenon |a to removerInfo(informação)
Bind phenomenon |b to especialidade
Bind phenomenon |c to removerEspecialidade(especialidade, unidade de saude)
Bind phenomenon |d to especialidade removida

Compose aspect AspRemoverInfo with Administrar informação
Bind domain |Máquina to Máquina de administração de informação
Bind domain |DominioA to Administrador Unidades Saúde
Bind domain |DominioB to Doenças
Bind phenomenon |a to removerInfo(informação)
Bind phenomenon |b to doença
Bind phenomenon |c to removerDoença(doença)
Bind phenomenon |d to doença removida

Compose aspect AspRemoverInfo with Administrar informação
Bind domain |Máquina to Máquina de administração de informação
Bind domain |DominioA to Administrador Unidades Saúde
Bind domain |DominioB to Campanhas de Vacinação
Bind phenomenon |a to removerInfo(informação)
Bind phenomenon |b to campanha
Bind phenomenon |c to removerCV(campanha)
Bind phenomenon |d to campanha removida
```

Figura C.47: Composição de AspRemoverInfo a Administrar informação.



a: AUS!{login(), logout(), inserirInfo(informação), actualizarInfo(informação), removerInfo(informação)}

b: AUS!{utilizador, palavra chave, especialidade, doença, campanha de vacinação}

c: MAI!{inserirEspecialidade(especialidade, unidade de saúde), actualizarEspecialidade(especialidade, unidade de saúde), removerEspecialidade(especialidade, unidade de saúde)}

d: US!{especialidade inserida, especialidade removida, especialidade actualizada}

e: MAI!{inserirDoença(doença), actualizarDoença(doença), removerDoença(doença)}

f: US!{doença inserida, doença removida, doença actualizada}

g: MAI!{inserirCV(campanha), actualizarCV(campanha), removerCV(campanha)}

h: US!{campanha inserida, campanha, removida, campanha actualizada}

i: MAI!{login(utilizador, palavra chave), logout(utilizador)}

j: U!{Login efectuado, Logout efectuado}

Figura C.48: Diagrama de problema Administrar informação composto.

```
Compose aspect AspLogin with Administrar sistema
Bind domain |Máquina to Máquina de administração de sistema
Bind Bind domain |Dominio to Administrador sistema
```

Figura C.49: Composição de AspLogin a Administrar sistema.

```
Compose aspect AspLogout with Administrar sistema
Bind domain |Máquina to Máquina de administração de sistema
Bind Bind domain |Dominio to Administrador sistema
```

Figura C.50: Composição de AspLogout a Administrar sistema.

```
Compose aspect AspInserirInfo with Administrar sistema
Bind domain |Máquina to Máquina de administração de sistema,
Bind domain |DominioA to Administrador sistema
Bind domain |DominioB to Unidades de Saúde
Bind phenomenon |a to inserirInfo(informação)
Bind phenomenon |b to unidade de saúde
Bind phenomenon |c to inserirUS(unidade de saude)
Bind phenomenon |d to unidade de saúde inserida
```

```
Compose aspect AspInserirInfo with Administrar sistema
Bind domain |Máquina to Máquina de administração de sistema
Bind domain |DominioA to Administrador sistema
Bind domain |DominioB to Reclamações
Bind phenomenon |a to inserirInfo(informação)
Bind phenomenon |b to tipo de reclamação
Bind phenomenon |c to inserirTipoReclamação(tipo)
Bind phenomenon |d to tipo inserido
```

Figura C.51: Composição de AspInserirInfo a Administrar sistema.

```
Compose aspect AspActualizarInfo with Administrar sistema
Bind domain |Máquina to Máquina de administração de sistema
Bind domain |DominioA to Administrador sistema
Bind domain |DominioB to Unidades de Saúde
Bind phenomenon |a to actualizarInfo(informação)
Bind phenomenon |b to unidade de saúde
Bind phenomenon |c to actualizarUS(unidade de saude)
Bind phenomenon |d to unidade de saúde actualizada

Compose aspect AspActualizarInfo with Administrar sistema
Bind domain |Máquina to Máquina de administração de sistema
Bind domain |DominioA to Administrador sistema
Bind domain |DominioB to Reclamações
Bind phenomenon |a to actualizarInfo(informação)
Bind phenomenon |b to tipo de reclamação
Bind phenomenon |c to actualizarTipoReclamação(tipo)
Bind phenomenon |d to tipo actualizado
```

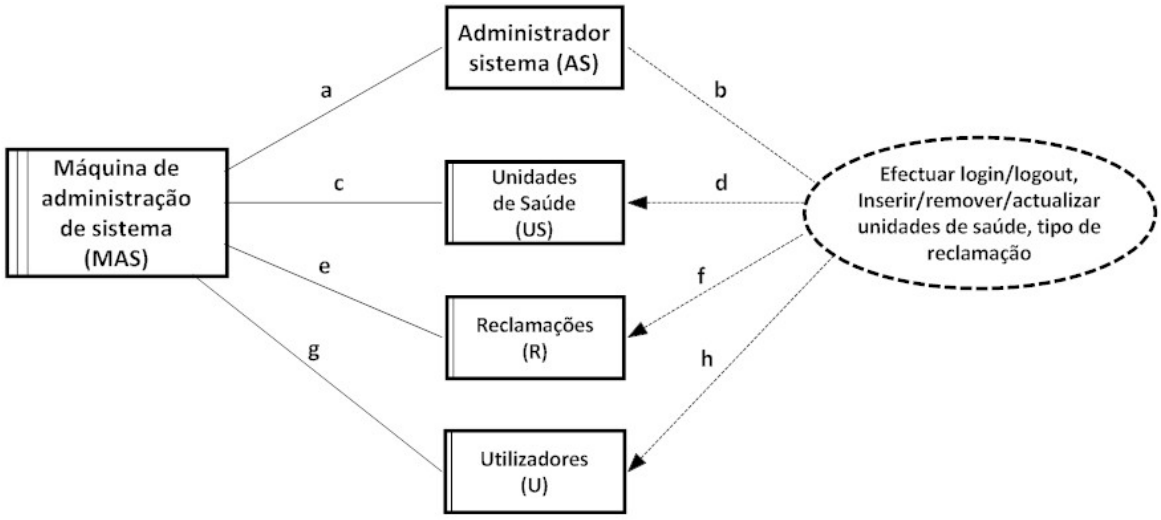
Figura C.52: Composição de AspActualizarInfo a Administrar sistema.

```
Compose aspect AspRemoverInfo with Administrar sistema
Bind domain |Máquina to Máquina de administração de sistema
Bind domain |DominioA to Administrador sistema
Bind domain |DominioB to Unidades de Saúde
Bind phenomenon |a to removerInfo(informação)
Bind phenomenon |b to unidade de saúde
Bind phenomenon |c to removerUS(unidade de saude)
Bind phenomenon |d to unidade de saúde removida

Compose aspect AspRemoverInfo with Administrar sistema
Bind domain |Máquina to Máquina de administração de sistema
Bind domain |DominioA to Administrador sistema
Bind domain |DominioB to Reclamações
Bind phenomenon |a to removerInfo(informação)
Bind phenomenon |b to tipo de reclamação
```

```
Bind phenomenon |c to removerTipoReclamação(tipo)
Bind phenomenon |d to tipo removido
```

Figura C.53: Composição de AspRemoverInfo a Administrar sistema.



a: AS!{login(), logout(), inserirInfo(informação), actualizarInfo(informação), removerInfo(informação)}
b: AS!{utilizador, palavra chave, unidade de saúde, tipo de reclamação}
c: MAS!{inserirUS(unidade de saúde), actualizarUS(unidade de saúde), removerUS(unidade de saúde)}
d: US!{unidade de saúde inserida, unidade de saúde removida, unidade de saúde actualizada}
e: MAS!{inserirTipoReclamação(tipo), actualizarTipoReclamação(tipo), removerTipoReclamação(tipo)}
f: R!{tipo inserido, tipo removido, tipo actualizado}
g: MAS!{login(utilizador, palavra chave), logout(utilizador)}
h: U!{Login efectuado, Logout efectuado}

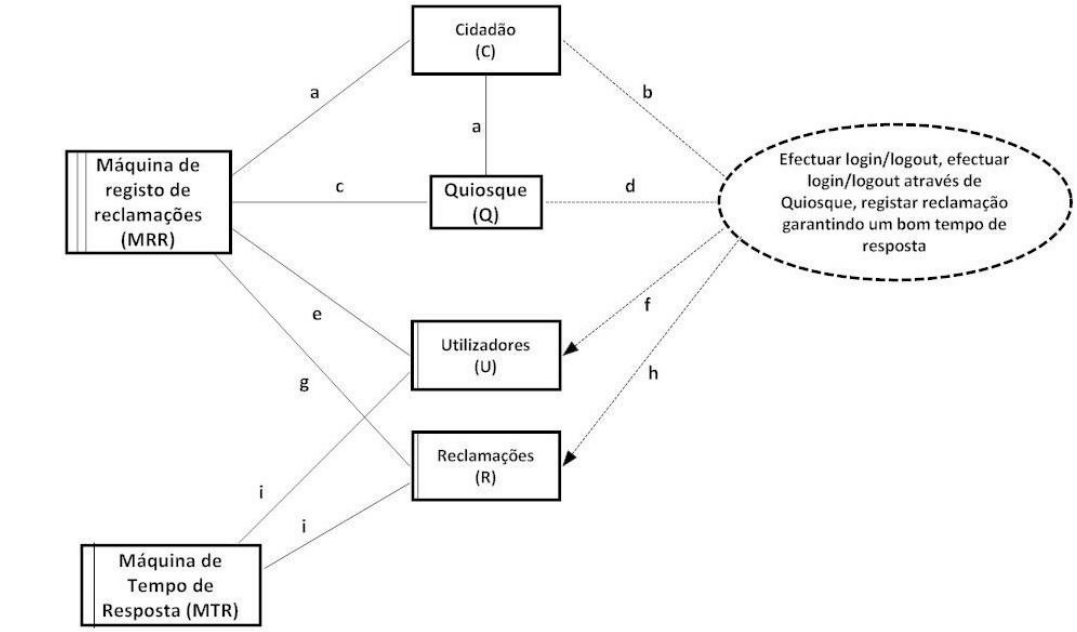
Figura C.54: Diagrama de problema Administrar sistema composto.

Diagramas de problema compostos com aspectos não funcionais

```
Compose aspect Tempo de Resposta with Registar Reclamação
Bind domain |Dominio to Reclamações

Compose aspect Tempo de Resposta with Registar Reclamação
Bind domain |Dominio to Utilizadores
```

Figura C.55: Composição de Tempo de Resposta a Registar reclamação.



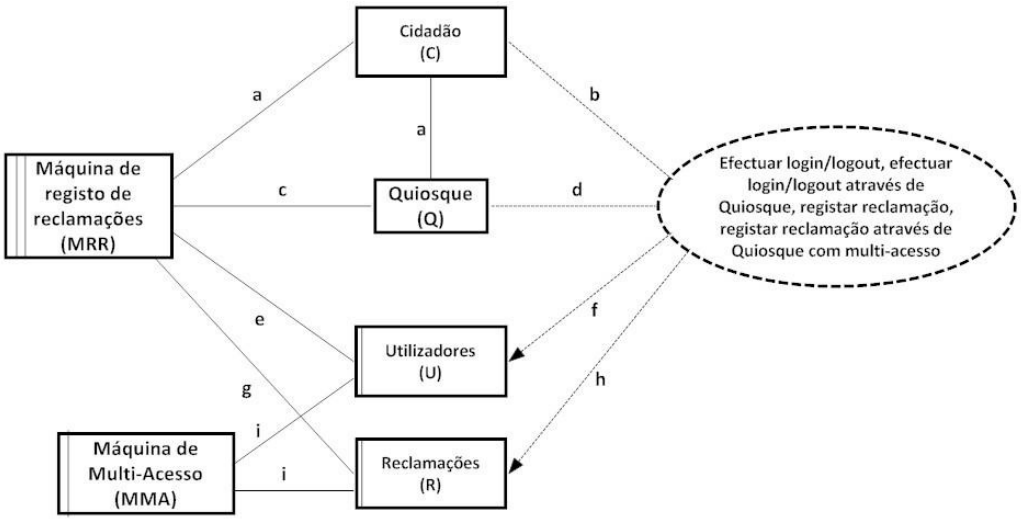
a: C!{login(), logout(), registarReclamação()}
b: C!{utilizador, palavra chave, reclamação}
c: Q!{login(), logout(), registarReclamação()}
d: Q!{utilizador, palavra chave, reclamação}
e: MRR!{login(utilizador, palavra chave), logout(utilizador)}
f: U!{login efectuado, , Logout efectuado, tempo de resposta garantido}
g: MRR!{registarReclamação(reclamação)}
h: R!{reclamação registada, tempo de resposta garantido}
i: MTR!{garantirTResposta()}

Figura C.56: Diagrama de problema Registar reclamação composto com Tempo de Resposta.

```
Compose aspect Multi-acesso with Registar Reclamação
Bind domain |Dominio to Utilizadores

Compose aspect Multi-acesso with Registar Reclamação
Bind domain |Dominio to Reclamações
```

Figura C.57: Composição de Multi-acesso a Registar reclamação.



a: C!{login(), logout(), registarReclamação()}
b: C!{utilizador, palavra chave, reclamação}
c: Q!{login(), logout(), registarReclamação()}
d: Q!{utilizador, palavra chave, reclamação}
e: MRR!{login(utilizador, palavra chave), logout(utilizador)}
f: U!{Login efectuado, Logout efectuado, multi-acesso permitido}
g: MRR!{registarReclamação(reclamação)}
h: R!{reclamação registada, multi-acesso permitido}
i: MMA!{permitirMA()}

Figura C.58: Diagrama de problema Registar reclamação composto com Multi-acesso.

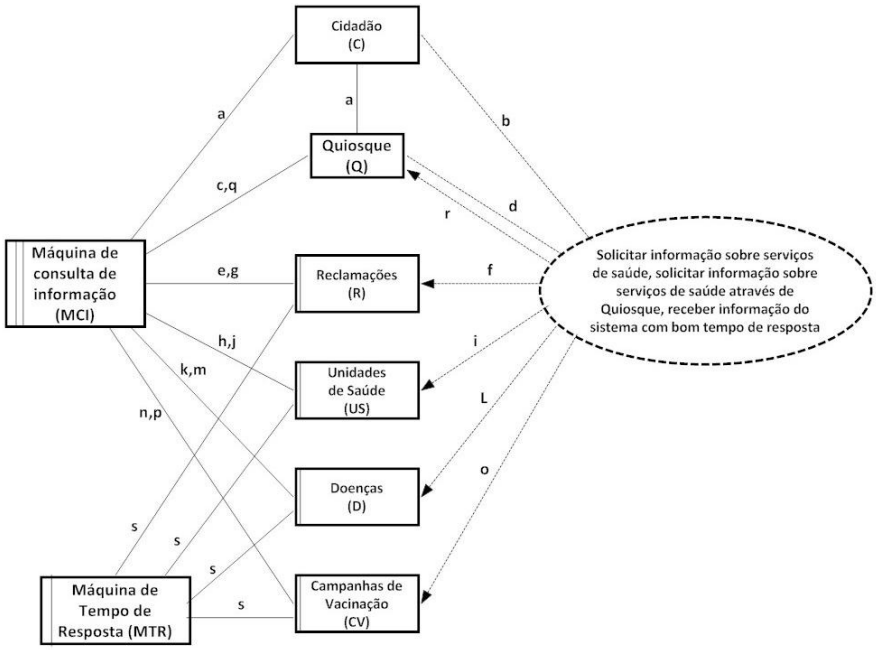
```
Compose aspect Tempo de Resposta with Solicitar Informação
Bind domain |Dominio to Reclamações

Compose aspect Tempo de Resposta with Solicitar Informação
Bind domain |Dominio to Unidades de Saúde

Compose aspect Tempo de Resposta with Solicitar Informação
Bind domain |Dominio to Doenças

Compose aspect Tempo de Resposta with Solicitar Informação
Bind domain |Dominio to Campanhas de Vacinação
```

Figura C.59: Composição de Tempo de Resposta a Solicitar informação.



a: C!{informação(tipo)}
b: C!{tipo da informação}
c: Q!{informação(tipo)}
d: Q!{tipo da informação}
e: MCI!{consultarReclamação(reclamação)}
f: R!{reclamação existente, tempo de resposta garantido}
g: R!{reclamação}
h: MCI!{consultarUS(especialidade), consultarUS(unidadeSaúde)}
i: US!{unidade saúde existente, especialidade existente, tempo de resposta garantido}
j: US!{unidade saúde, especialidade}
k: MCI!{consultarDoença(doença)}
L: D!{doença existente, tempo de resposta garantido}
m: D!{doença}
n: MCI!{consultarVacinação(campanha)}
o: CV!{campanha vacinação existente, , tempo de resposta garantido}
p: CV!{campanha vacinação}
q: MCI!{receberInformação(informação)}
r: Q!{informação recebida}
s: MTR!{garantirTResposta()}

Figura C.60: Diagrama de problema Solicitar informação composto com Tempo de Resposta.

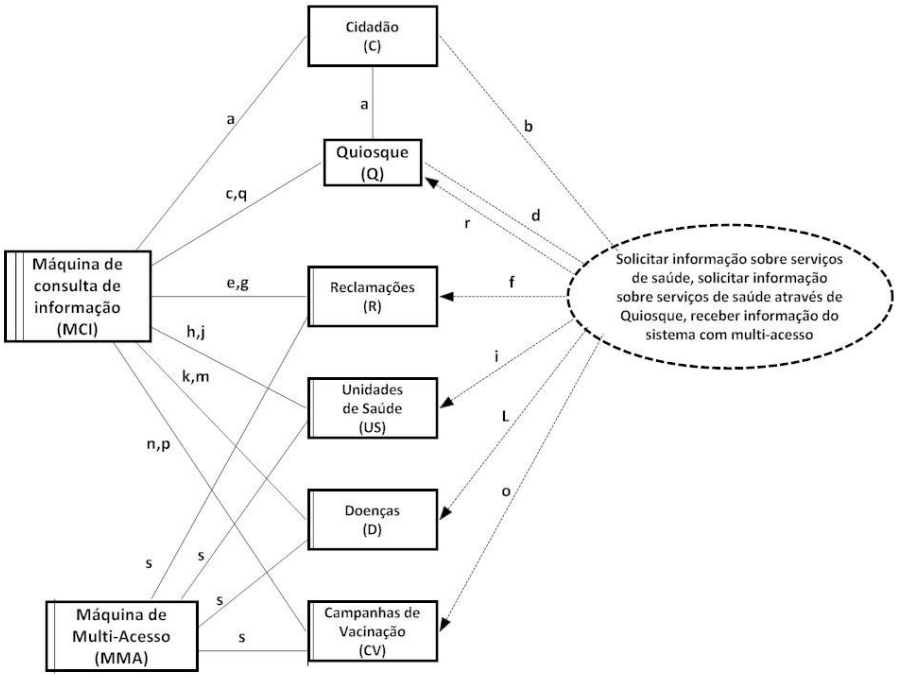
```
Compose aspect Multi-acesso with Solicitar Informação
Bind domain |Dominio to Unidades de Saúde

Compose aspect Multi-acesso with Solicitar Informação
Bind domain |Dominio to Doenças

Compose aspect Multi-acesso with Solicitar Informação
Bind domain |Dominio to Campanhas de Vacinação

Compose aspect Multi-acesso with Solicitar Informação
Bind domain |Dominio to Reclamações
```

Figura C.61: Composição de Multi-acesso a Solicitar informação.

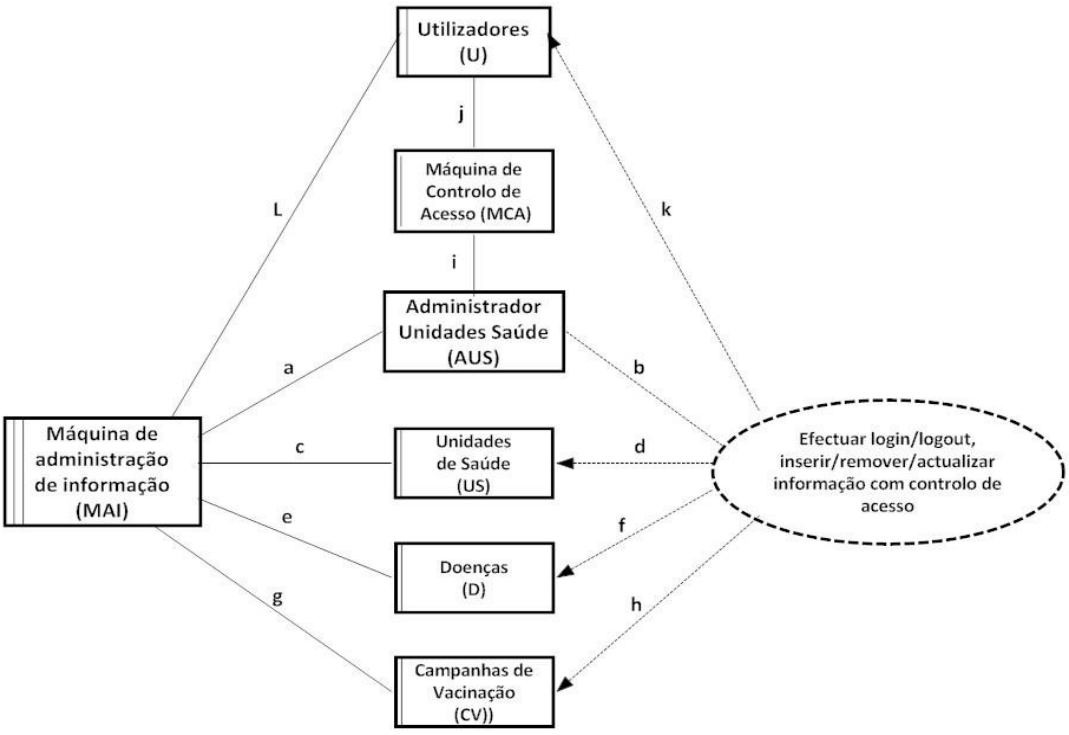


a: C!{informação(tipo)}
b: C!{tipo da informação}
c: Q!{informação(tipo)}
d: Q!{tipo da informação}
e: MCI!{consultarReclamação(reclamação)}
f: R!{reclamação existente, multi-acesso permitido}
g: R!{reclamação}
h: MCI!{consultarUS(especialidade), consultarUS(unidadeSaúde)}
i: US!{unidade saúde existente, especialidade existente, multi-acesso permitido}
j: US!{unidade saúde, especialidade}
k: MCI!{consultarDoença(doença)}
L: D!{doença existente, multi-acesso permitido}
m: D!{doença}
n: MCI!{consultarVacinação(campanha)}
o: CV!{campanha vacinação existente, multi-acesso permitido}
p: CV!{campanha vacinação}
q: MCI!{receberInformação(informação)}
r: Q!{informação recebida}
s: MMA!{permitirMA()}

Figura C.62: Diagrama de problema Solicitar informação composto com Multi-acesso.

```
Compose aspect Segurança with Administrar informação
Bind domain |Dominio to Administrador Unidades Saúde
```

Figura C.63: Composição de Segurança a Administrar informação.



a: AUS!{login(), logout(), inserirInfo(informação), actualizarInfo(informação), removerInfo(informação)}

b: AUS!{especialidade, doença, campanha de vacinação, utilizador, palavra chave}

c: MAI!{inserirEspecialidade(especialidade, unidade de saúde), removerEspecialidade(especialidade, unidade de saúde), removerEspecialidade(especialidade, unidade de saúde)}

d: US!{especialidade inserida, especialidade removida, especialidade actualizada}

e: MAI!{inserirDoença(doença), removerDoença(doença), removerDoença(doença)}

f: US!{doença inserida, doença removida, doença actualizada}

g: MAI!{inserirCV(campanha), removerCV(campanha), removerCV(campanha)}

h: US!{campanha inserida, campanha, removida, campanha actualizada}

i: AUS!{login()}

j: MCA!{verificarPalavra(utilizador, palavra-chave)}

k: U!{Login efectuado, Logout efectuado, acesso permitido}

L: MAI!{login(utilizador, palavra chave), logout(utilizador)}

Figura C.64: Diagrama de problema Administrar informação composto com Segurança.

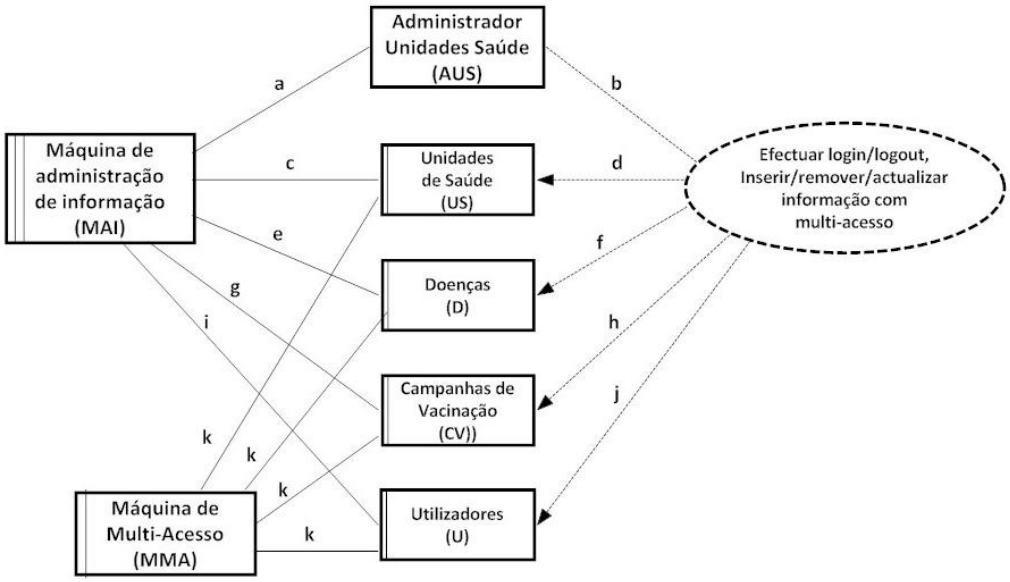
```
Compose aspect Multi-acesso with Administrar informação
Bind domain |Dominio to Utilizadores

Compose aspect Multi-acesso with Administrar informação
Bind domain |Dominio to Unidades de Saúde

Compose aspect Multi-acesso with Administrar informação
Bind domain |Dominio to Doenças

Compose aspect Multi-acesso with Administrar informação
Bind domain |Dominio to Campanhas de Vacinação
```

Figura C.65: Composição de Multi-acesso a Administrar informação.



a: AUS!{login(), logout(), inserirInfo(informação), actualizarInfo(informação), removerInfo(informação)}

b: AUS!{utilizador, palavra chave, especialidade, doença, campanha de vacinação}

c: MAI!{inserirEspecialidade(especialidade, unidade de saúde), actualizarEspecialidade(especialidade, unidade de saúde), removerEspecialidade(especialidade, unidade de saúde)}

d: US!{especialidade inserida, especialidade removida, especialidade actualizada, multi-acesso permitido}

e: MAI!{inserirDoença(doença), actualizarDoença(doença), removerDoença(doença)}

f: US!{doença inserida, doença removida, doença actualizada, multi-acesso permitido}

g: MAI!{inserirCV(campanha), actualizarCV(campanha), removerCV(campanha)}

h: US!{campanha inserida, campanha, removida, campanha actualizada, multi-acesso permitido}

i: MAI!{login(utilizador, palavra chave), logout(utilizador)}

j: U!{Login efectuado, Logout efectuado, multi-acesso permitido}

k: MMA!{permitirMA()}

Figura C.66: Diagrama de problema Administrar informação composto com Multi-acesso.

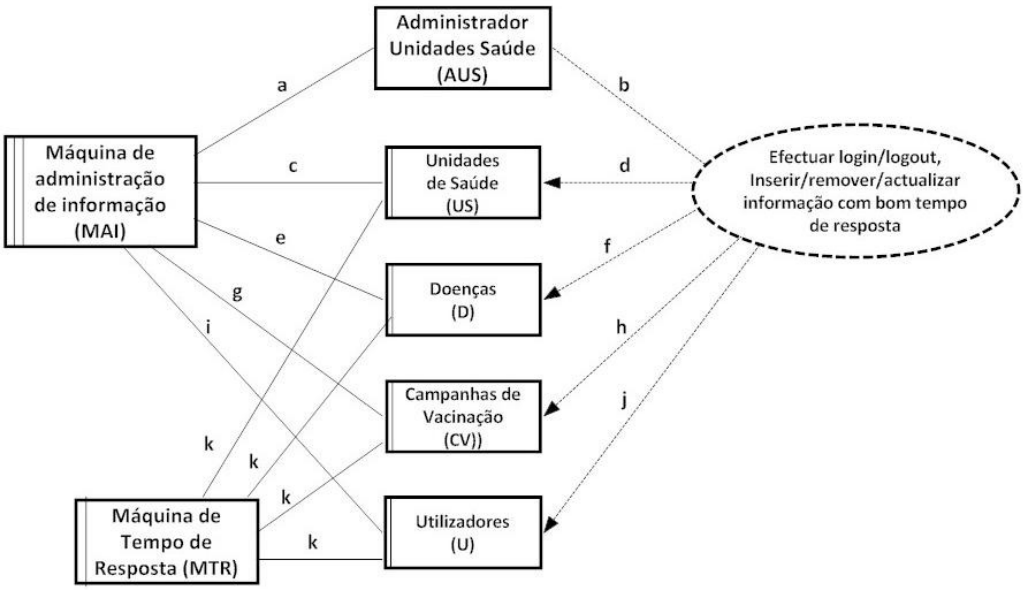
```
Compose aspect Tempo de Resposta with Administrar informação
Bind domain |Dominio to Utilizadores

Compose aspect Tempo de Resposta with Administrar informação
Bind domain |Dominio to Unidades de Saúde

Compose aspect Tempo de Resposta with Administrar informação
Bind domain |Dominio to Doenças

Compose aspect Tempo de Resposta with Administrar informação
Bind domain |Dominio to Campanhas de Vacinação
```

Figura C.67: Composição de Tempo de Resposta a Administrar informação.



a: AUS!{login(), logout(), inserirInfo(informação), actualizarInfo(informação), removerInfo(informação)}

b: AUS!{utilizador, palavra chave, especialidade, doença, campanha de vacinação}

c: MAI!{inserirEspecialidade(especialidade, unidade de saúde), actualizarEspecialidade(especialidade, unidade de saúde), removerEspecialidade(especialidade, unidade de saúde)}

d: US!{especialidade inserida, especialidade removida, especialidade actualizada, tempo de resposta garantido}

e: MAI!{inserirDoença(doença), actualizarDoença(doença), removerDoença(doença)}

f: US!{doença inserida, doença removida, doença actualizada, tempo de resposta garantido}

g: MAI!{inserirCV(campanha), actualizarCV(campanha), removerCV(campanha)}

h: US!{campanha inserida, campanha, removida, campanha actualizada, tempo de resposta garantido}

i: MAI!{login(utilizador, palavra chave), logout(utilizador)}

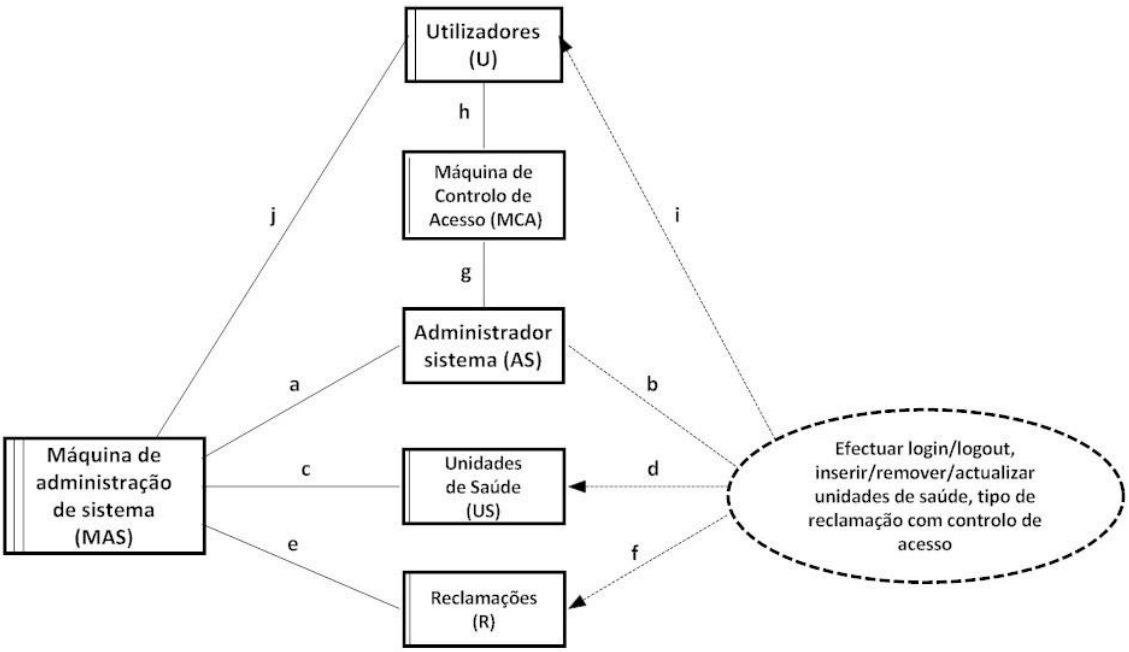
j: U!{Login efectuado, Logout efectuado, tempo de resposta garantido}

k: MTR!{garantirTResposta()}

Figura C.68: Diagrama de problema Administrar informação composto com Tempo de Resposta.

```
Compose aspect Segurança with Administrar sistema
Bind domain |Dominio to Administrador sistema
```

Figura C.69: Composição de Segurança a Administrar sistema.



a: AS!{login(), logout(), inserirInfo(informação), actualizarInfo(informação), removerInfo(informação)}
b: AS!{unidade de saúde, tipo de reclamação, utilizador, palavra chave}
c: MAS!{inserirUS(unidade de saúde), removerUS(unidade de saúde), removerUS(unidade de saúde)}
d: US!{unidade de saúde inserida, unidade de saúde removida, unidade de saúde actualizada}
e: MAS!{inserirTipoReclamação(tipo), removerTipoReclamação(tipo), removerTipoReclamação(tipo)}
f: R!{tipo inserido, tipo removido, tipo actualizado}
g: AS!{login()}
h: MCA!{verificarPalavra(utilizador, palavra-chave)}
i: U!{Login efectuado, Logout efectuado, acesso permitido}
j: MAS!{login(utilizador, palavra chave), logout(utilizador)}

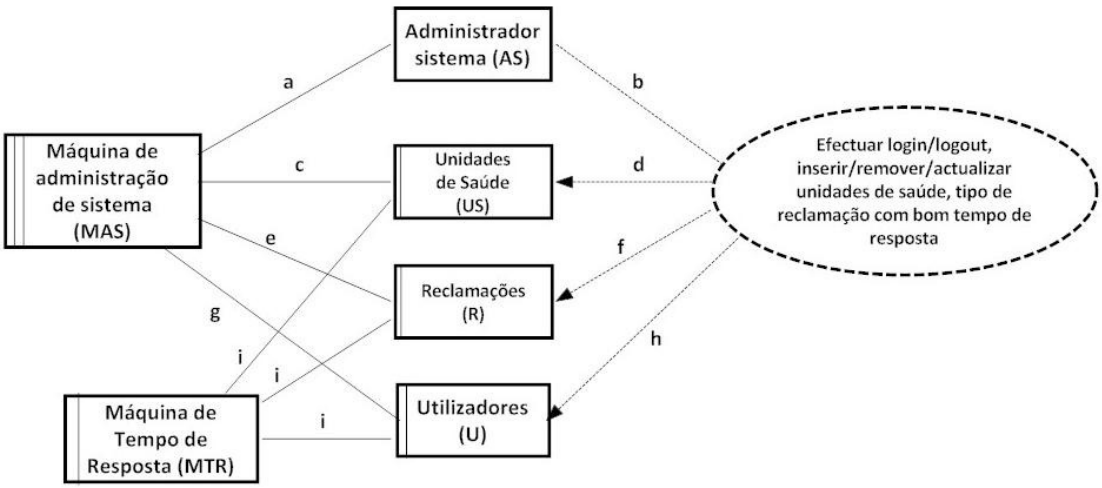
Figura C.70: Diagrama de problema Administrar sistema composto com Segurança.

```
Compose aspect Tempo de Resposta with Administrar sistema
Bind domain |Dominio to Unidades de Saúde

Compose aspect Tempo de Resposta with Administrar sistema
Bind domain |Dominio to Reclamações

Compose aspect Tempo de Resposta with Administrar sistema
Bind domain |Dominio to Utilizadores
```

Figura C.71: Composição de Tempo de Resposta a Administrar sistema.



a: AS!{login(), logout(), inserirInfo(informação), actualizarInfo(informação), removerInfo(informação)}
b: AS!{unidade de saúde, tipo de reclamação, utilizador, palavra chave}
c: MAS!{inserirUS(unidade de saúde), removerUS(unidade de saúde), removerUS(unidade de saúde)}
d: US!{unidade de saúde inserida, unidade de saúde removida, unidade de saúde actualizada, tempo de resposta garantido}
e: MAS!{inserirTipoReclamação(tipo), removerTipoReclamação(tipo), removerTipoReclamação(tipo)}
f: R!{tipo inserido, tipo removido, tipo actualizado, tempo de resposta garantido}
g: MAS!{login(utilizador, palavra chave), logout(utilizador)}
h: U!{Login efectuado, Logout efectuado, tempo de resposta garantido}
i: MTR!{garantirTResposta()}

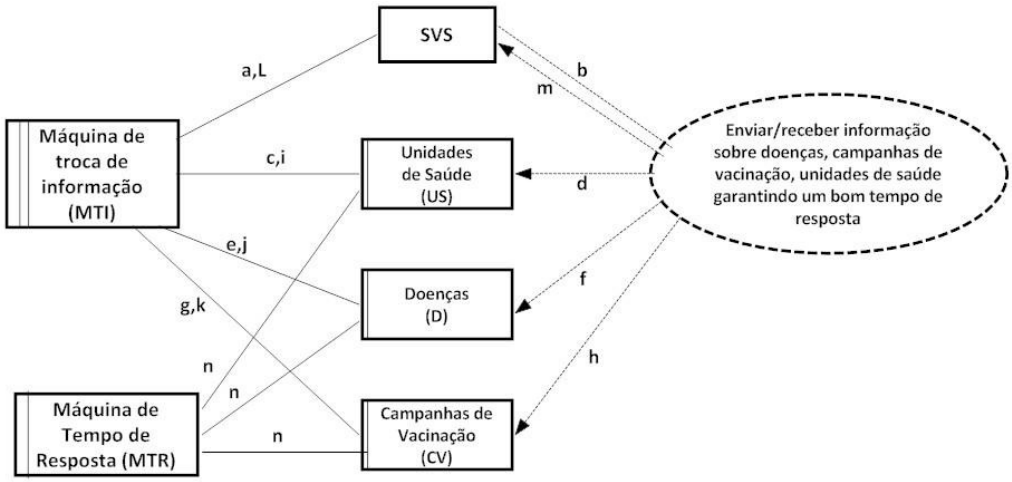
Figura C.72: Diagrama de problema Administrar sistema composto com Tempo de Resposta.

```
Compose aspect Tempo de Resposta with Trocar Informação
Bind domain |Dominio to Unidades de Saúde

Compose aspect Tempo de Resposta with Trocar Informação
Bind domain |Dominio to Doenças

Compose aspect Tempo de Resposta with Trocar Informação
Bind domain |Dominio to Campanhas de Vacinação
```

Figura C.73: Composição de Tempo de Resposta a Trocar informação.



a: SVS!{informação(tipo), informação enviada com sucesso}
b: SVS!{informação, tipo da informação}
c: MTI!{actualizarUS(informação)}
d: US!{informação actualizada, tempo de resposta garantido}
e: MTI!{actualizarDoenças(informação)}
f: D!{informação actualizada, tempo de resposta garantido}
g: MTI!{actualizarCV(informação)}
h: CV!{informação actualizada, tempo de resposta garantido}
i: US!{informação}
j: D!{informação}
k: CV!{informação}
L: MTI!{enviarInformação(informação)}
m: SVS!{informação enviada}
n: MTR!{garantirTResposta()}

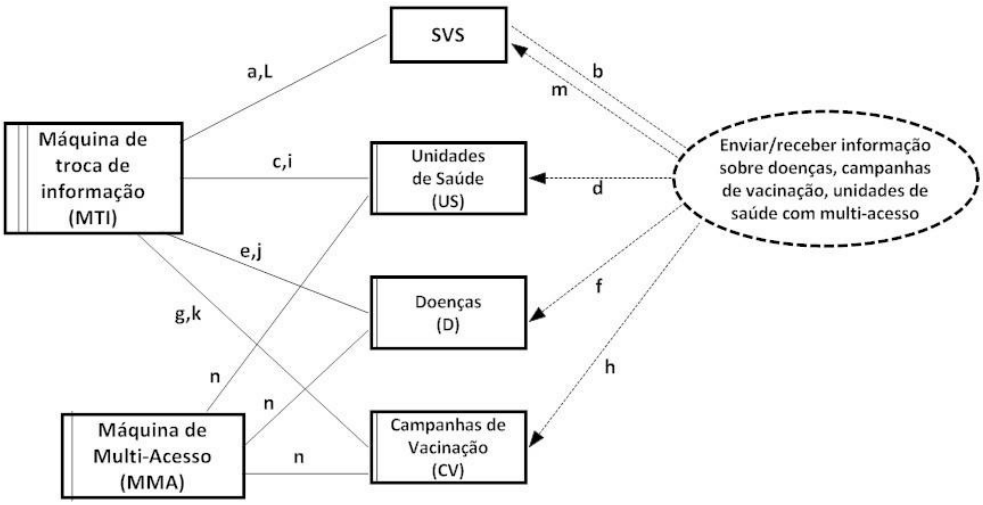
Figura C.74: Diagrama de problema Trocar informação composto com Tempo de Resposta.

```
Compose aspect Multi-acesso with Trocar Informação
Bind domain |Dominio to Unidades de Saúde

Compose aspect Multi-acesso with Trocar Informação
Bind domain |Dominio to Doenças

Compose aspect Multi-acesso with Trocar Informação
Bind domain |Dominio to Campanhas de Vacinação
```

Figura C.75: Composição de Multi-acesso a Trocar informação.

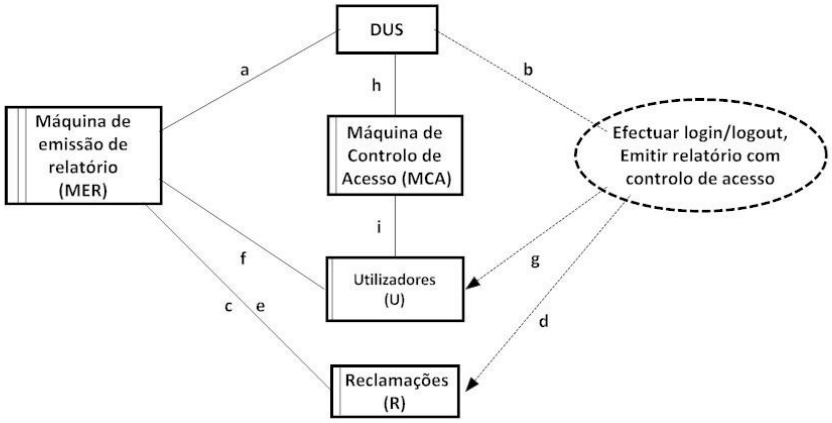


a: SVS!{informação(tipo), informação enviada com sucesso}
b: SVS!{informação, tipo da informação}
c: MTI!{actualizarUS(informação)}
d: US!{informação actualizada, multi-acesso permitido}
e: MTI!{actualizarDoenças(informação)}
f: D!{informação actualizada, multi-acesso permitido}
g: MTI!{actualizarCV(informação)}
h: CV!{informação actualizada, multi-acesso permitido}
i: US!{informação}
j: D!{informação}
k: CV!{informação}
L: MTI!{enviarInformação(informação)}
m: SVS!{informação enviada}
n: MMA!{permitirMA()}

Figura C.76: Diagrama de problema Trocar informação composto com Multi-acesso.

```
Compose aspect Segurança with Emitir relatório
Bind domain |Dominio to Director Unidade de Saúde
```

Figura C.77: Composição de Segurança a Emitir relatório.



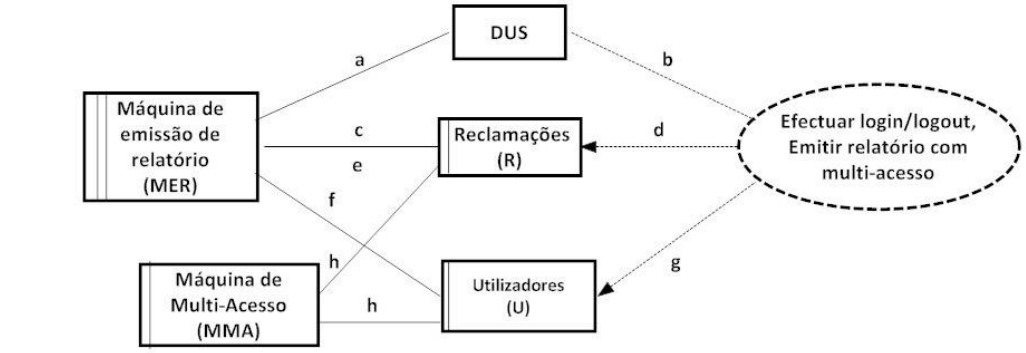
a: DUS!{login(), logout(), emitirRelatório()}
b: DUS!{utilizador, palavra chave, emitirRelatório()}
c: MER!{emitirRelatório(relatório)}
d: R!{relatório emitido}
e: R!{relatório}
f: MER!{login(utilizador, palavra chave), logout(utilizador)}
g: U!{Login efectuado, Logout efectuado, acesso permitido}
h: C!{login()}
i: MCA!{verificarPalavra(utilizador, palavra-chave)}

Figura C.78: Diagrama de problema Emitir relatório composto com Segurança.

```
Compose aspect Multi-acesso with Emitir relatório
Bind domain |Dominio to Utilizadores

Compose aspect Multi-acesso with Emitir relatório
Bind domain |Dominio to Reclamações
```

Figura C.79: Composição de Multi-acesso a Emitir relatório.



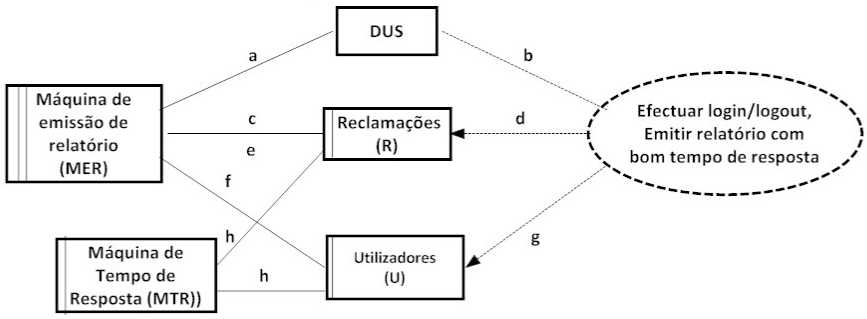
a: DUS!{login(), logout(), emitirRelatório()}
b: DUS!{utilizador, palavra chave, emitirRelatório()}
c: MER!{emitirRelatório(relatório)}
d: R!{relatório emitido, multi-acesso permitido}
e: R!{relatório}
f: MER!{login(utilizador, palavra chave), logout(utilizador)}
g: U!{Login efectuado, Logout efectuado, multi-acesso permitido}
h: MMA!{permitirMA()}

Figura C.80: Diagrama de problema Emitir relatório composto com Multi-acesso.

```
Compose aspect Tempo de Resposta with Emitir relatório
Bind domain |Dominio to Utilizadores

Compose aspect Tempo de Resposta with Emitir relatório
Bind domain |Dominio to Reclamações
```

Figura C.81: Composição de Tempo de Resposta a Emitir relatório.

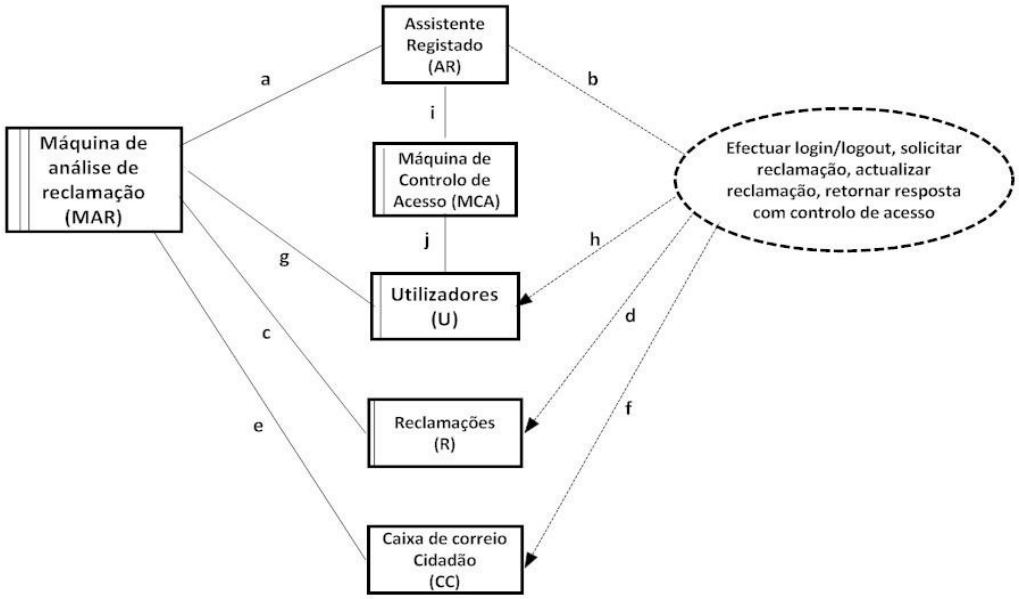


a: DUS!{login(), logout(), emitirRelatório()}
b: DUS!{utilizador, palavra chave, emitirRelatório()}
c: MER!{emitirRelatório(relatório)}
d: R!{relatório emitido, tempo de resposta garantido}
e: R!{relatório}
f: MER!{login(utilizador, palavra chave), logout(utilizador)}
g: U!{Login efectuado, Logout efectuado, tempo de resposta garantido}
h: MTR!{garantirTResposta()}

Figura C.82: Diagrama de problema Emitir relatório composto com Tempo de Resposta.

```
Compose aspect Segurança with Analisar Reclamação
Bind domain |Dominio to Assistente Registado
```

Figura C.83: Composição de Segurança a Analisar reclamação.



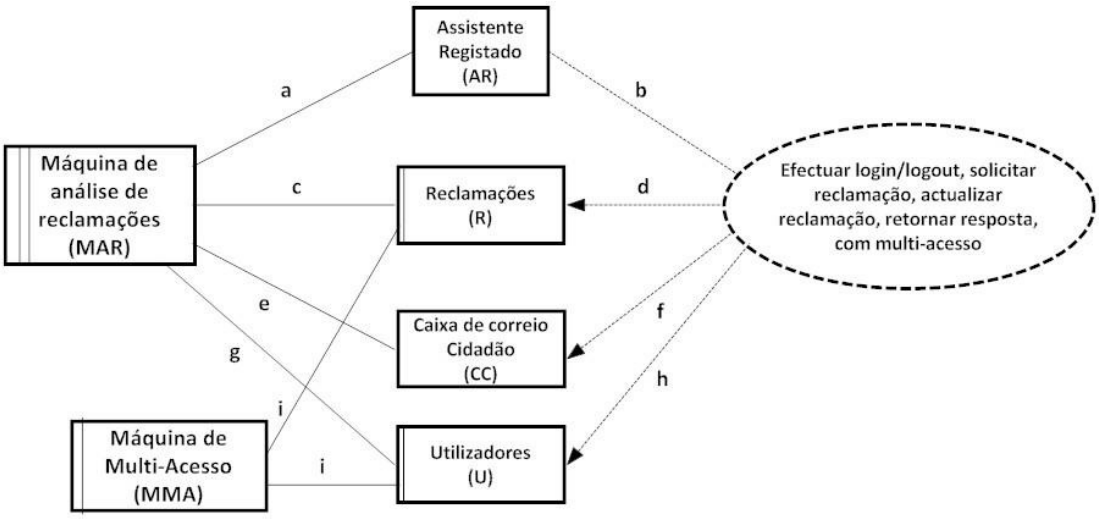
a: AR!{login(), logout(), obterReclamação(), actualizarReclamação(reclamação), enviarResposta(reclamação)}
b: AR!{reclamação, dados da análise, resposta da reclamação, utilizador, palavra chave, obterReclamação()}
c: MAR!{obterReclamação(), actualizarReclamação(reclamação)}
d: R!{reclamação obtida, reclamação actualizada}
e: MAR!{enviarResposta(reclamação, cidadão)}
f: CC!{reclamação enviada}
g: MAR!{login(utilizador, palavra chave), logout(utilizador)}
h: U!{Login efectuado, Logout efectuado, acesso permitido}
i: AR!{login()}
j: MCA!{verificarPalavra(utilizador, palavra-chave)}

Figura C.84: Diagrama de problema Analisar reclamação composto com Segurança.

```
Compose aspect Multi-acesso with Analisar Reclamação
Bind domain |Dominio to Utilizadores

Compose aspect Multi-acesso with Analisar Reclamação
Bind domain |Dominio to Reclamações
```

Figura C.85: Composição de Multi-acesso a Analisar reclamação.



a: AR!{login(), logout(), obterReclamação(), actualizarReclamação(reclamação), enviarResposta(reclamação)}

b: AR!{utilizador, palavra chave, reclamação, dados da análise, resposta da reclamação, obterReclamação()}

c: MAR!{obterReclamação(), actualizarReclamação(reclamação)}

d: R!{reclamação obtida, reclamação actualizada, multi-acesso permitido}

e: MAR!{enviarResposta(reclamação, cidadão)}

f: CC!{reclamação enviada}

g: MAR!{login(utilizador, palavra chave), logout(utilizador)}

h: U!{Login efectuado, Logout efectuado, multi-acesso permitido}

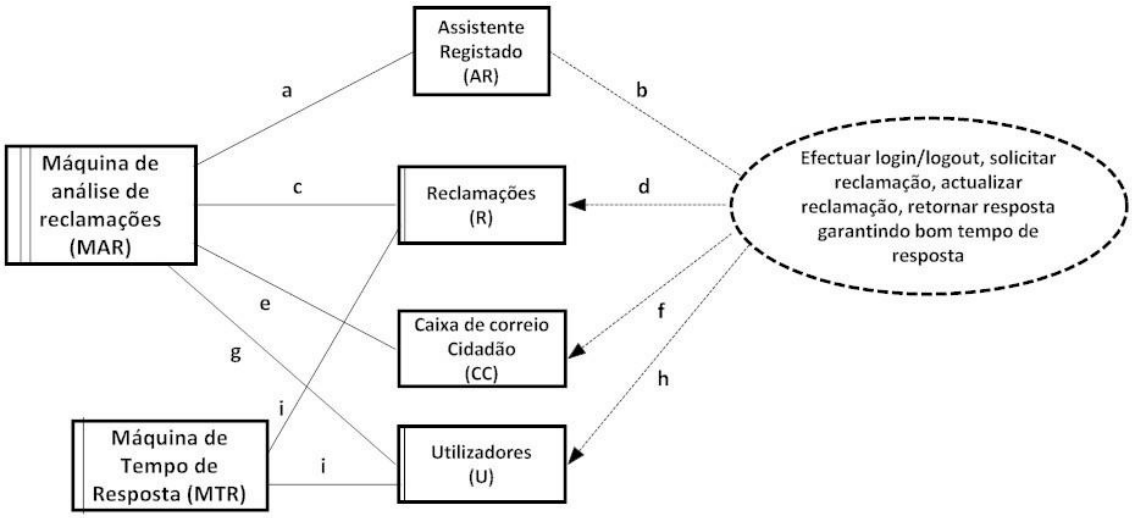
i: MMA!{permitirMA()}

Figura C.86: Diagrama de problema Analisar reclamação composto com Multi-acesso.

```
Compose aspect Tempo de Resposta with Analisar Reclamação
Bind domain |Dominio to Utilizadores

Compose aspect Tempo de Resposta with Analisar Reclamação
Bind domain |Dominio to Reclamações
```

Figura C.87: Composição de Tempo de Resposta a Analisar reclamação.



a: AR!{login(), logout(), obterReclamação(), actualizarReclamação(reclamação), enviarResposta(reclamação)}

b: AR!{utilizador, palavra chave, reclamação, dados da análise, resposta da reclamação, obterReclamação()}

c: MAR!{obterReclamação(), actualizarReclamação(reclamação)}

d: R!{reclamação obtida, reclamação actualizada, tempo de resposta garantido}

e: MAR!{enviarResposta(reclamação, cidadão)}

f: CC!{reclamação enviada}

g: MAR!{login(utilizador, palavra chave), logout(utilizador)}

h: U!{Login efectuado, Logout efectuado, tempo de resposta garantido}

i: MTR!{garantirTResposta()}

Figura C.88: Diagrama de problema Analisar reclamação composto com Tempo de Resposta.